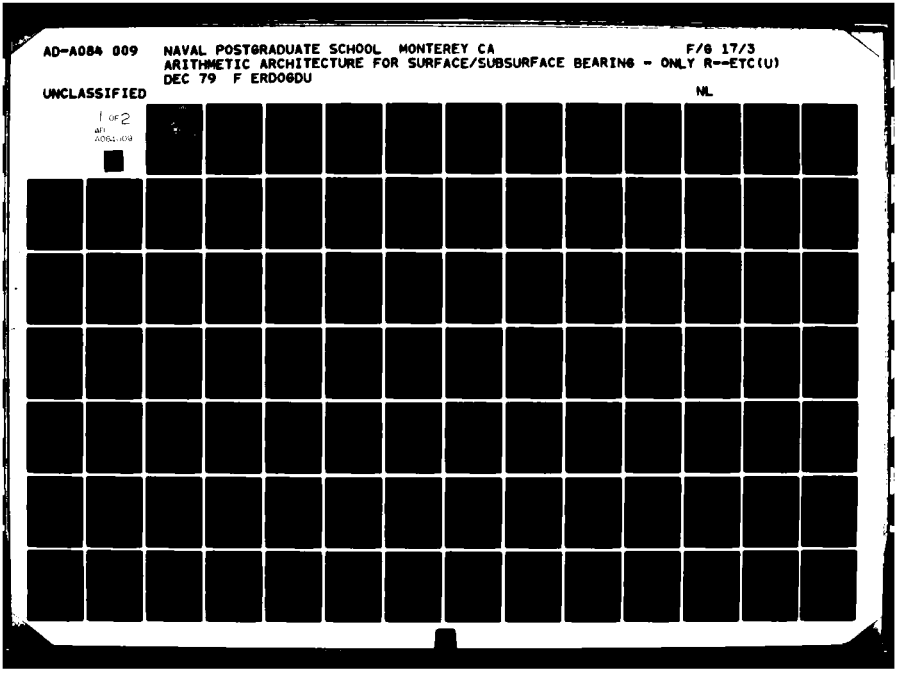
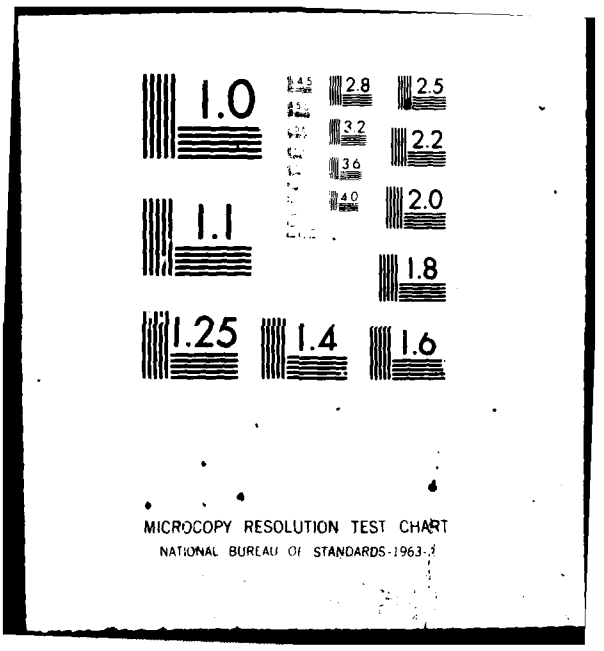
AD-A084 009 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 17/3
ARITHMETIC ARCHITECTURE FOR SURFACE/SUBSURFACE BEARING - ONLY R--ETC(U)
DEC 79 F ERDOGDU

UNCLASSIFIED NL

1 OF 2
AD
A084009




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(2) LEVEL II

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

(6)

ARITHMETIC ARCHITECTURE FOR SURFACE/SUBSURFACE BEARING - ONLY RADAR TRACKING BY MICROCOMPUTER.

by

Fatih/Erdogdu

(11) December 1979

Thesis Advisor: M.L. Cotton

Approved for public release; distribution unlimited.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A084009	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Arithmetic Architecture for Surface/ Subsurface Bearing - Only Radar Tracking by Microcomputer	5. TYPE OF REPORT & PERIOD COVERED Master's Thesis;
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Fatih Erdogdu	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940	12. REPORT DATE December 1979
	13. NUMBER OF PAGES 177
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Bearing-Only Radar Tracking
Microprocessor
Radar Tracking by Microcomputer

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In most shipboard CIC's true motion plots of own ship and/or surface/subsurface contact motions are performed with the aid of electromechanical devices. These are

DD FORM 1 JAN 73 1473 (Page 1)

EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(20. ABSTRACT Continued)

inaccurate and have a variety of drawbacks. This thesis describes a fast, labor-saving, bearing only radar tracking solution utilizing various microprocessors.

Approved for public release; distribution unlimited.

Arithmetic Architecture for Surface/Subsurface Bearing - Only Radar Tracking by Microcomputer

by

Fatih Erdogdu
Lieutenant, Turkish Navy
B.S., Naval Postgraduate School, 1978

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

December 1979

Author _____

Approved by: _____
Thesis Advisor

_____
Second Reader

_____
Chairman, Department of Electrical Engineering

_____
Dean of Science and Engineering

ABSTRACT

In most shipboard CIC's true motion plots of own ship and/or surface/subsurface contact motions are performed with the aid of electromechanical devices. These are inaccurate and have a variety of drawbacks.

This thesis describes a fast, labor-saving, bearing only radar tracking solution utilizing various microprocessors.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

Passive localization and tracking techniques are of interest in a variety of microcomputer based surface/ subsurface radar applications.

Because a significant portion of the cost of building or maintaining a warship is the electronics in its sensor and weapon system, and because a great amount of time and manpower is employed in performing simple but important tasks, microcomputers offer the potential to:

(1) Reduce the cost of digital systems,

(2) Perform some complex functions at remote stations, relieving the congestion at larger central computer facilities, and

(3) Perform functions currently handled by watch personnel, thus reducing the manning requirements of watch sections.

An example of this is the problem of manual tracking of radar contacts and the solution of Maneuvering Board Problems.

It is the purpose of this study to demonstrate than an alternate approach to the solution of the problems mentioned above can be developed and implemented by using a micro-computer system, while at the same time maintaining a human engineered user interface.

II. FORMULATION OF THE PROBLEM

A. PREFACE

For Naval ships not equipped with Naval Tactical Data Systems (NTDS), operations performed by the Combat Information Center (CIC) during a normal peacetime watch include manual tracking of radar contacts and solution of maneuvering board problems. The environment emulated by the Intel Intellec Microcomputer Development System (MDS) provides a data subset of the NTDS. The subset of information is provided via radar interface to the MDS System.

The interface between the microcomputer and the surface radar was not constructed to due cost, time and manpower constraints.

The scenario of the study was limited to the development of a Kalman Filter bearing-only tracking algorithm for the Intel Intellec MDS microcomputer system and Intel SBC-310 High-Speed Mathematics Unit Performance. Simulation of the circuitry is necessary to interface with the AN/SPS-10 surface search radar.

The parameters for the AN/SPS-10 surface search radar were used for computations requiring specific radar parameters. This radar was chosen because all information regarding the radar is available at the Naval Postgraduate School.

A Kalman filter was selected because it minimizes the estimation error in a well defined statistical sense and most complex calculations of the tracking filters typically used for bearing-only radar tracking that is easily computable with a microcomputer system.

The system input/output parameters are shown in Figure 1.

B. TYPICAL COMBAT INFORMATION CENTER OPERATIONS

During normal peacetime steaming, the CIC watch team may consist of from two to ten or even more personnel, depending on the size of the ship as well as on the complexity of the equipment being used.

Among the problems that are normally solved by CIC personnel, special mention needs to be made of those of plotting contacts and the determination of parameters such as course, speed and closest point of approach (CPA) of those contacts. This is a tedious and error-prone task; it often requires most of the time and effort of the CIC team and it is vitally important to the safety of the ship. This had led to the installation of equipment to reduce the amount of workload in the CIC while at the same time improving the reliability of the constant information provided to the bridge. This equipment includes dead reckoning devices and NC2 plotter.

1. Maneuvering Board Plotting Sheets

The maneuvering Board Plotting Sheets (H.O. 2665-10) have been prepared in order to facilitate the solution of a ship's relative movement problem.

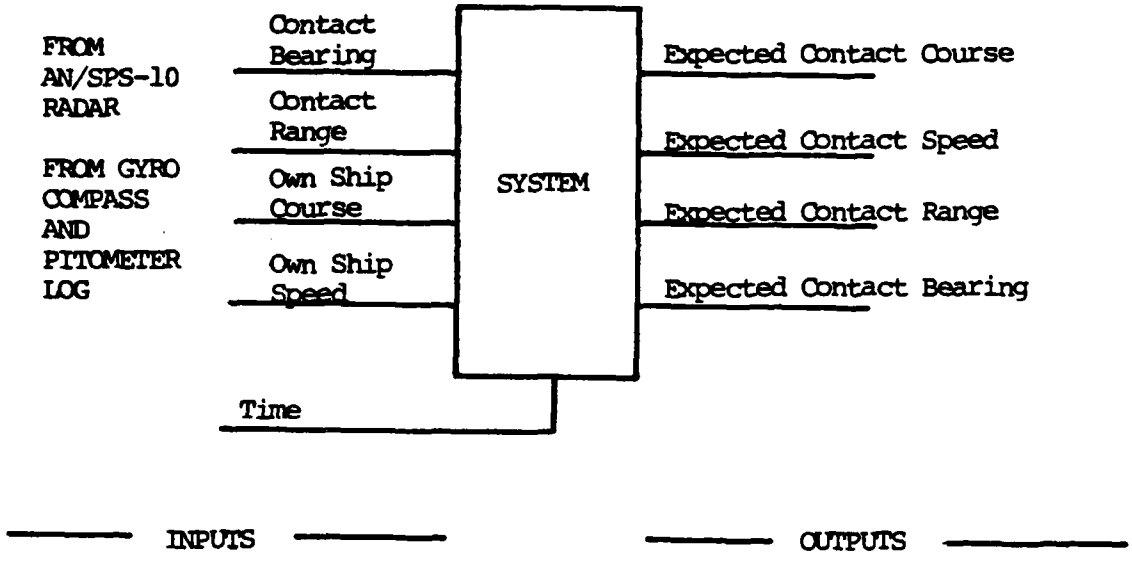


Figure 1. SYSTEM INPUT/OUTPUT PARAMETERS

The primary responsibility of CIC is to provide information and recommendations on the tactical situations, such as contact course, speed, CPA information, are defined using the "Maneuvering Board" plotting sheets. But the use of the Maneuvering Board requires some pracice, also it requires the complete attention of one person during the CIC operation.

2. Dead-Reckoning Equipment

This equipment maintains a continuous, up to the minute, geographic plot of own ship's plot in the CIC.

The Dead-Reckoning System consists of the following basic components: (1) Dead-Reckoning Analyzer (DRA); (2) Dead-Reckoning Indicator (DRI); and (3) Dead-Reckoning Time (DRT). Course and speed inputs of own ship are fed into the DRA from Gyrocompass and pitometer-log and then to the DRT, where they cause a movable source of light to trace the ship-track continuously.

In all systems as mentioned above the target's relative position measurements are obtained from radar repeater and plotted on the systems which are read to be used. The range measurement accuracy is at best ± 20 yards and bearing measurement accuracy is ± 1 degrees with a well calibrated repeater and a well trained operator. The own ship speed and course are used to determine the own ship's velocity vector. The course and speed of own ship may vary ± 1 degree and ± 1 knot depending on the helmsman, weather

conditions and the ship instruction. The combined result and the manual solution will cause ± 5 degrees in course and ± 3 knots in speed of the target information. Also the major disadvantage of the manual solution, in addition to its poor accuracy, is the time required to obtain the solution and the inability to obtain solutions of either the target or own ship maneuvers between radar position measurements, typically 3 or more minutes apart.

C. MICROPROCESSOR TECHNOLOGY

Recent advances in large scale integration (LSI) semiconductor process technology have made possible substantial reductions in the cost and size of digital logic circuits. Microprocessors represent a very remarkable achievement of engineering ingenuity and industrial know-how at their best.

Since 1973, the year in which Intel Corporation shipped the first 8080, 8-bit N-channel microprocessor, a number of manufacturers have developed similar products and because of this competition prices have been drastically lowered. With hardware and software system development costs as they are, much micro work is done on a custom basis. The primary advantage of microcomputer based systems is to have specialized and dedicated equipment solving specific problems.

The microprocessor families are: (1) 4-bit machine; (2) 8-bit machine; (3) 16-bit machine; and (4) bit-slices

architecture. 16-bit microprocessor is the best solution for our concern of tracking problem. But Intel's 8080 was selected as an 8-bit microprocessor and programmed for double precision arithmetic to perform Kalman Filter algorithms, because of its lower cost and availability at the Naval Postgraduate School with all its hardware and software support devices.

D. SCENARIO

In order for a ship to maintain operational readiness as a unit of a task force, it must be aware of the current operational environment. The operational environment is defined as the surface/subsurface contact profiles in the geographic area of interest. The contact profile consists of friendly, hostile, and unknown contacts with associative contact characteristics. Contact characteristics are such measurements from radar, range and bearing to accurately determine the course and speed of a radar contact. If the radar measurements were perfectly accurate and the target had zero acceleration, then the tracking problem would be velocity vector problem as shown in Figure 2, velocity vector and contact relative motion plot as shown in Figure 3. But the radar bearing and range measurements on any scan have errors which are a function of the radar system and the received signal to noise ratio. The measurement errors will be discussed after defining the Kalman Filter parameters.

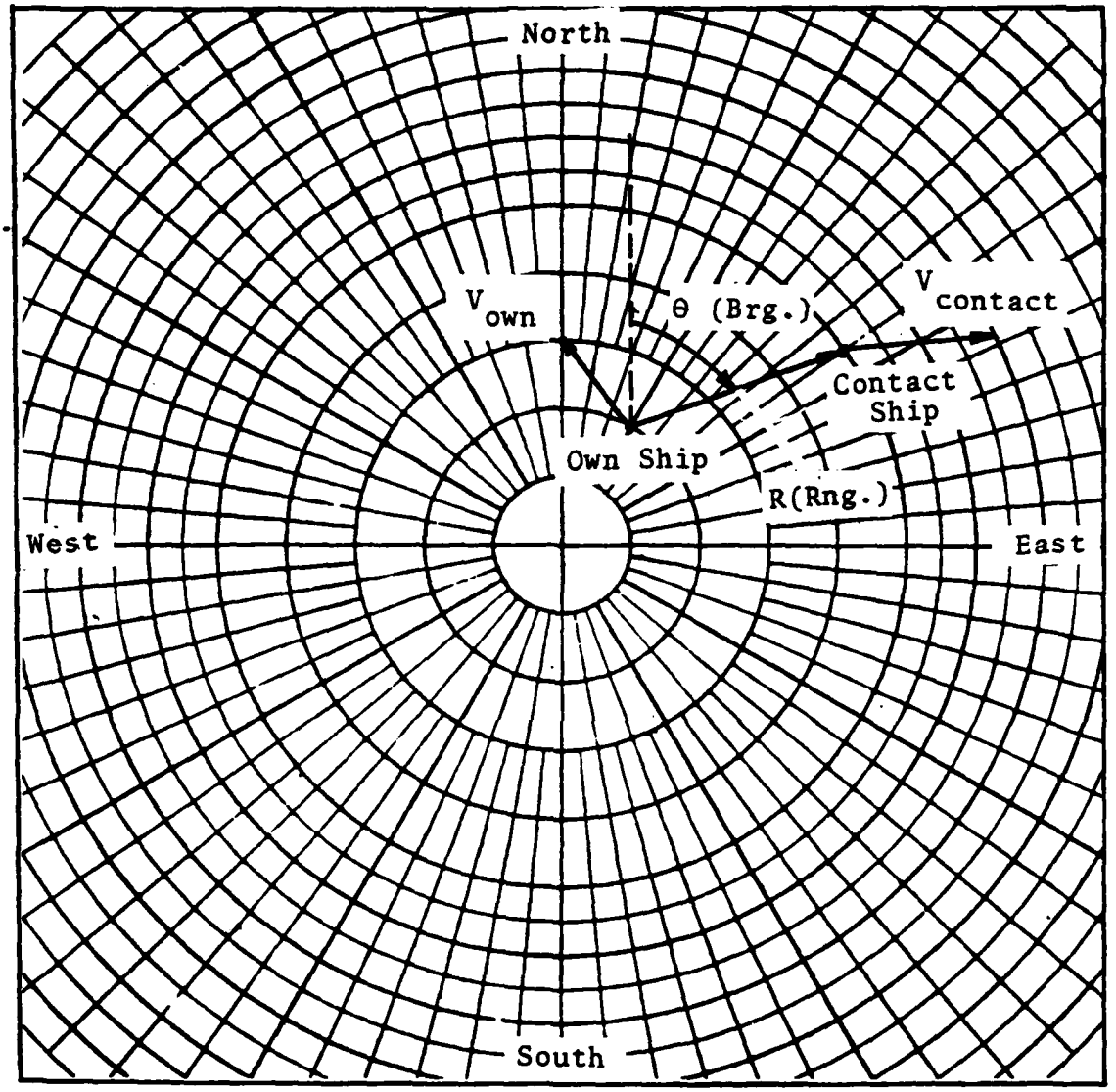


Figure 2. GEOGRAPHIC PLOT OF CONTACT AND OWN SHIP MOTION

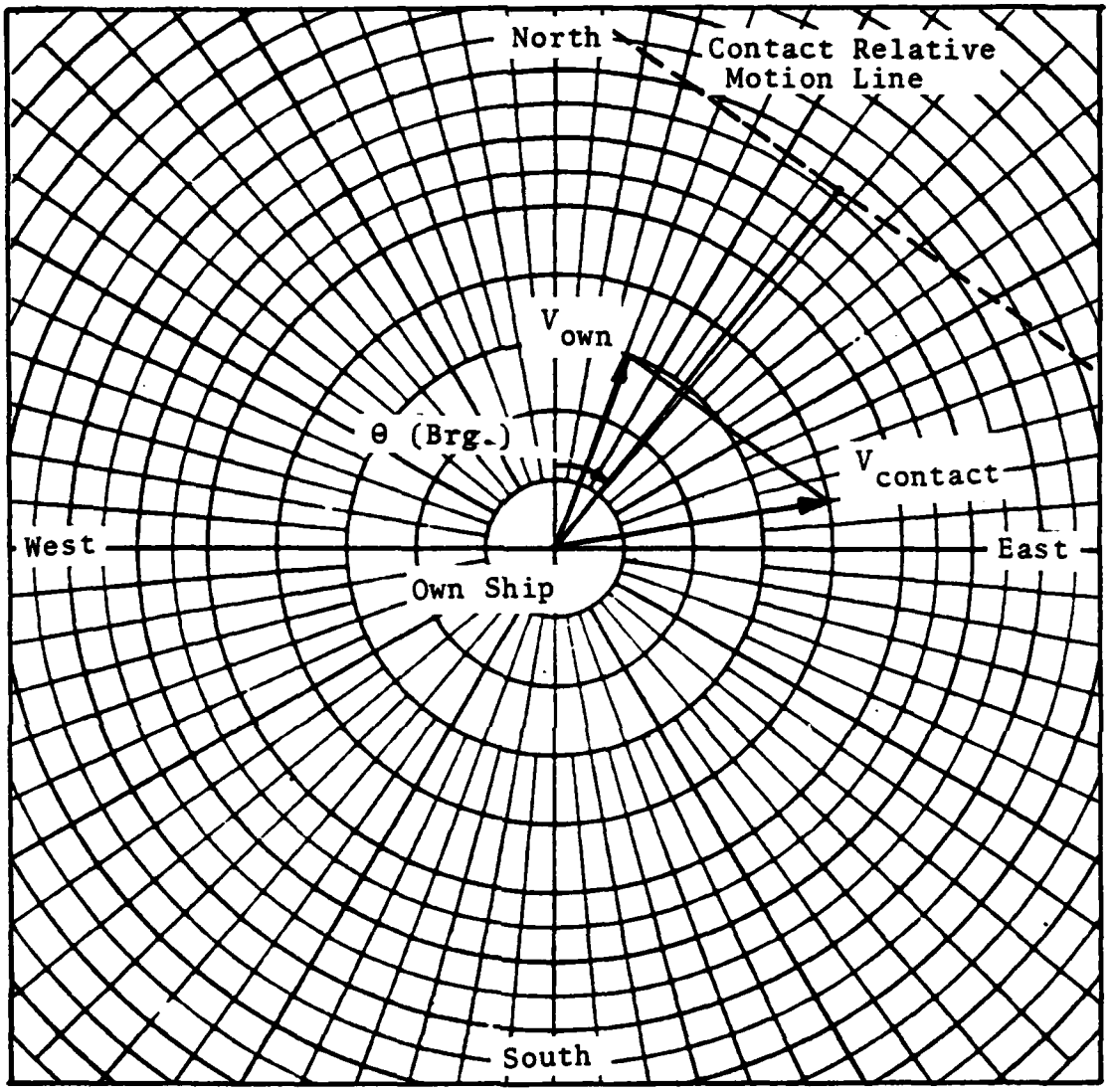


Figure 3. VELOCITY VECTOR AND CONTACT RELATIVE MOTION PLOT
(Own Ship assumed at the origin.)

The contact motion on a X,Y coordinate system can be approximated by linear dynamic system. The target dynamics can be represented in a matrix form as shown below to define the sample data system such as a search radar where the measurements are taken once each antenna rotation (Reference 13)

$$Z_{n+1} = \Phi Z_n + \Gamma a_n$$

where:

$$Z_n = \begin{vmatrix} x_n \\ \dot{x}_n \\ y_n \\ \dot{y}_n \end{vmatrix}, \quad \Phi = \begin{vmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{vmatrix}, \quad \Gamma = \begin{vmatrix} T^2/2 & 0 \\ T & 0 \\ 0 & T^2/2 \\ 0 & T \end{vmatrix}$$

and,

Z_n = vehicle state vector at scan n,

x_n = E/W position of the vehicle,

$\dot{x}_n$ = E/W velocity of the vehicle,

y_n = N/S position of the vehicle,

$\dot{y}_n$ = N/S velocity of the vehicle,

T = interval between observations,

a_n = acceleration acting upon the vehicle from scan (n) to (n+1).

A Kalman filter is a common optimal filtering technique for estimating the state of a linear system. It is selected to implement the tracking problem because it is computationally the most demanding technique in our concern. The details of the Kalman filter will be analyzed in Chapter IV.

III. SYSTEM DESIGN CONSIDERATIONS

A. HARDWARE DESIGN CONSIDERATIONS

Microprocessors, in general, are not complete computers, but Central Processor Units (CPU) implemented with one to ten large-scale integrated-circuit chips. Large-scale integration (LSI) chips are comprised of 1,000 or more gates; many LSI chips hold over 6,000 gates or a "complete central processor."

The word "micro-computer" refers to a small stored program computer comprising memory and input/output circuits together with a microprocessor CPU.

Interface design and programming make the mass-produced microcomputers into new special-purpose dedicated machines.

The Intellec Microcomputer Development System (MDS) is a complete, coordinated computer system designed around Intel's 8080 microprocessor. The MDS has a 2 microsecond instruction cycle, a repertoire of 72 powerful instructions, unlimited subroutine nesting, and a versatile interrupt scheme.

The MDS was selected because of its 8080 microprocessor, its memory capabilities, its interrupt mechanism, and its extended I/O capabilities. Also, the selection of the Intel Microcomputer Development System (MDS) was made for this theis because of its immediate availability at the Naval Postgraduate School.

A Datamedia Elite 2500 Video Terminal was chosen to present alphanumeric information because its features include: (1) Editing and roll operation modes; (2) 50 to 3600 baud programmable speed transmission; (3) protected fields, (4) computer derived or high light field (blink); (5) addressable cursor.

Because of the computation of Kalman filter algorithms, floating-point calculation required, an SBC-310 high-speed math unit, developed by Intel Corporation, was incorporated. In performing high speed mathematical functions, the Math Unit acts as an intelligent processor, performing a repertoire of 14 arithmetic functions and at least an oder of magnitude faster than comparable software routines. Why it is to selected to perform the algorithms will be discussed in Section C of this Chapter, and Figure 4 shows the final configuration of the equipment used in developing this thesis. Also, start-up procedures for the system were listed in Appendix B.

B. SOFTWARE DESIGN CONSIDERATIONS

1. Language Selection

Choice of the computer language to be used was partially dependent upon the hardware configuration. PL/M-80 was preferred as a programming language to be used, after selecting MDS hardware system, to the assembly language because of its ease as programming and immediate availability.

PL/M-80 is a high level language developed by Intel Corporation and esigned especially for system and applications

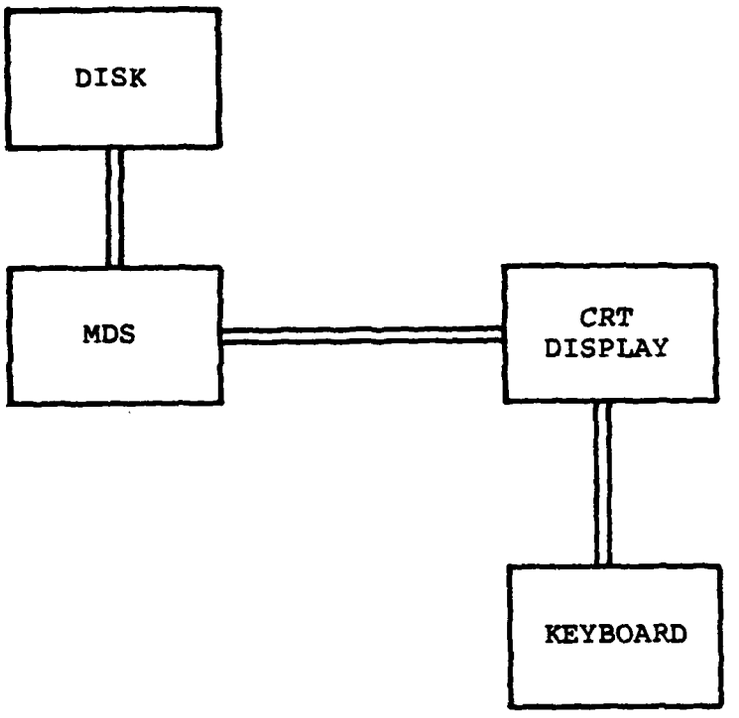


Figure 4. EQUIPMENT CONFIGURATION

programming for the Intel 8080 microprocessor. It is a block structured language which speeds the microprocessor program development by relieving the programmer of the tedious task of memory and register management.

ISIS-II disk operating system also developed by Intel Corporation for the Intellec MDS system has a resident PL/M-80 compiler which has been very useful during the

implementation, debugging and testing the program. The PL/M-80 compiler accepts the statements as input and produces a machine-code program module as output.

PL/M statements are divided into two basic categories:

1. DECLARE and PROCEDURE statements.
2. EXECUTABLE statements, which are all other than declare and procedure statements.

EXECUTABLE statements cause machine code to be generated. DECLARE statements cause variables to be defined associated with objects and PROCEDURE capability permits program to be constructed in a modular fashion, which has numerous advantages including efficiency of coding, readability of programs, ease of debugging and possibility of using the same procedure in more than one program (Ref. 6).

2. Floating Point Arithmetic

Floating point arithmetic is required because range of numbers to be represented is large and sometimes unpredictable, especially in the Kalman filter algorithm. In this algorithm when the time concern to use a floating point arithmetic is disadvantageous, because it needs additional subroutines or hardware, but when the accuracy concern we must use it in our programming.

In selecting floating-point formats, the primary considerations are word size and the radix of the arithmetic. In much of the literature, 24-bit word size is regarded as too small for scientific computation. The Intel standard for floating point arithmetic has been adopted to apply to

all general purpose products including software systems, and components. Such as FPAL (Floating-Point arithmetic library) for the 8080 and Math Board (SBC-310) made of series 3000 bit slices have already been produced using this standard.

As the floating-point arithmetic formats concern there are two standard formats which have been accepted from Intel (Ref. 12): (a) 32-bit short precision word; (b) 62-bit long precision word. The 32-bit short precision word format with binary radix was chosen because of the hardware consideration by selectin the MDS system and 8080 microprocessor with SBC-310 high speed mathematics unit was the fastest and available at the Naval Postgraduate School.

Comparison will be done next with its compatibles:

(1) AM9511 Arithmetic processing unit (Advanced Micro Devices, Inc.)

(2) 8080/8085 Floating-Point Arithmetic Library (FPAL) (Intel Corporation)

(3) 24-bit special floating point (Ref. 11)

(4) SBC 310 High Speed Mathematic Unit (Intel Corporation).

a. AM9511 Arithmetic Processing Unit

The AM9511 is a monolithic MOS/LSI device which provides high speed, 16 and 32 bit fixed and 32 bit floating-point arithmetic, plus a group of transcendental derived functions, control and conversion commands.

The AM9511 APU is contained within a single dual-in-line 24 pin chip as described in detail in Ref. 3.

The distinctive characteristics of the AM9511 APU include:

- Fixed-point 16 and 32 bit operation
- Floating-point 32 bit operation
- Binary data formats
- Basic add, subtract, multiply and divide
- Trigonometric sine, cosine, tangent
- Inverse trigonometric arcsine, arccosine, arctangent
- Square roots
- Logarithms base on 10 and e
- Exponentiation
- Float-to-fixed and fixed-to-float conversions
- stack oriented operand storage
- DMA or programmed I/O data transfers
- General purpose 8-bit data interface.

Data Formats:

All fixed-point operands and results are represented as binary two's complement integer values. The 16 bit format can express numbers with a range of -32,768 to +32,767. The 32 bit format can express numbers with a range of -2,147,483,648 to +2,147,483,647.

The floating point format uses a 32 bit word width as shown in Figure 5. The most significant bit (bit 31) indicates the sign of mantissa. The next seven bits form the exponent and remaining 24 bits form the mantissa value.

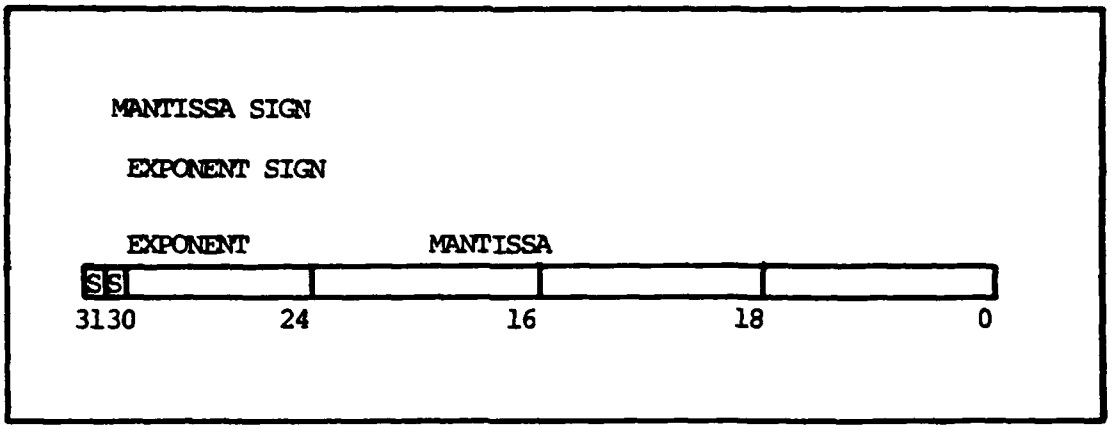


Figure 5. FLOATING-POINT FORMAT FOR AM9511 APU

The exponent of the base 2 is an unbiased two's complement number with a range of -64 to +63. The mantissa is a sign-magnitude number with an assumed binary point just to the left of the most significant mantissa bit (bit 2). All floating point values must be normalized which makes bit 23 always equal to 1 except when representing a value of zero. The number zero is represented with binary zeros in all 32 bit positions.

The operation of the APU is described in Ref. 3 in detail with the command formats and their descriptions. The AM9511 interested commands, its clock cycles and typical execution times for three different clock frequencies such as 2 MHz, 3 MHz and 4 MHz are shown in Table I.

COMMAND MNEMONIC	CLOCK CYCLES	TYPICAL EXECUTION TIMES		
		AM9511 (2 MHz)	AM9511-1 (3 MHz)	AM9511-4 (4 MHz)
32-BIT FIXED POINT OPERATIONS				
ADD (DADD)	22-22	10	7	5
SUBTRACT (DSUB)	38-40	19	13	10
MULTIPLY, LOWER (DMUL)	184-210	97	65	49
MULTIPLY, UPPER (DMUU)	182-218	91	61	46
DIVIDE (DDIV)	186-210	98	66	50
32-BIT FLOATING-POINT PRIMARY OPERATIONS				
ADD (FADD)	54-368	27	18	14
SUBTRACT (FSUB)	70-370	35	24	18
MULTIPLY (FMUL)	146-168	73	49	37
DIVIDE (FDIV)	154-184	77	52	38
32-BIT FLOATING-POINT DERIVED OPERATIONS				
SQUAREROOT (SQRT)	782-870	391	262	200
LOG BASE 10 (LOG)	4474-7132	2237	1493	1125
LOG BASE e (LN)	4298-6956	2149	1435	1098

Note: All time values are specified in microseconds.

Table I. AM9511 ARITHMETIC PROCESSING UNIT COMMAND EXECUTION TIMES

COMMAND MNEMONIC	CLOCK CYCLES	TYPICAL EXECUTION TIMES		
		AM9511 (2 MHz)	AM9511-1 (3 MHz)	AM9511-4 (4 MHz)
CONTINUATION OF 32-BIT FLOATING-POINT DERIVED OPERATIONS				
SINE (SIN)	3786-4808	1898	1266	950
COSINE (COS)	3840-4878	1920	1280	960
TANGENT (TAN)	4894-5886	2447	1635	1225
ARCSINE (ASIN)	6230-7938	3115	2080	1560
ARCCOSINE (ACOS)	6304-8284	3152	2105	1580
ARCTANGENT (ATAN)	4992-6536	2496	1665	1250
EXPONENT (EXP)	3784-4878	1887	1265	949
POWER (PWR)	8280-12032	4145	2765	2075
32-BIT DATA AND STACK MANIPULATION OPERATIONS				
FIXED-TO-FLOAT CONVERSION (FLTD)	56-342	28	19	14
FLOAT-TO-FIXED CONVERSION (FXTD)	20-336	45	30	23
FLOAT POINT SIGN CHANGE (CHSF)	16-20	8	6	4
EXCHANGE (XCHF)	26	13	9	7

Note: All time values are specified in microseconds.

Table 1 (Continuation). AM9511 ARITHMETIC PROCESSING UNIT COMMAND EXECUTION TIMES

b. 8080/8085 Floating-Point Arithmetic Library (FPAL)

The FPAL procedures reside in object code module form in the library "FPAL.LIB" on the Intellec ISIS-II System diskette. They are self contained and can be used in component, OEM-board, or Intellec Microcomputer, Development System. It contains basic floating point subroutines and functions (referred to generally as "procedures"). The operations provided are:

- Addition, subtraction, multiplication, division
- Value comparison
- Conversion between decimal and binary floating point and 32 bit signed integer formats
- A default error handler subroutine.

For all operations single precision format is used and in addition to these operations, a number of procedures are provided to deal with the Floating Point Record (FPR). This is a reserved, 18 byte work area used to collect status and error information, and as an accumulator for intermediate results. The procedures supporting the FPR perform FPR initialization, change error recovery options, check the contents of FPR fields, and pass numbers between the FPR and memory.

The FPAL also includes a default error-handler subroutine. This subroutine is called when an invalid number is used in a floating-point operation or if overflow, underflow, or division by zero are not handled by an arithmetic subroutine. We may also write our own error

handler, so long as it conforms to the formats described in Reference 2.

The FPAL can be used by assembly langugages or PL/M programs.

In general, the following steps must be observed to use the Floating-Point Library:

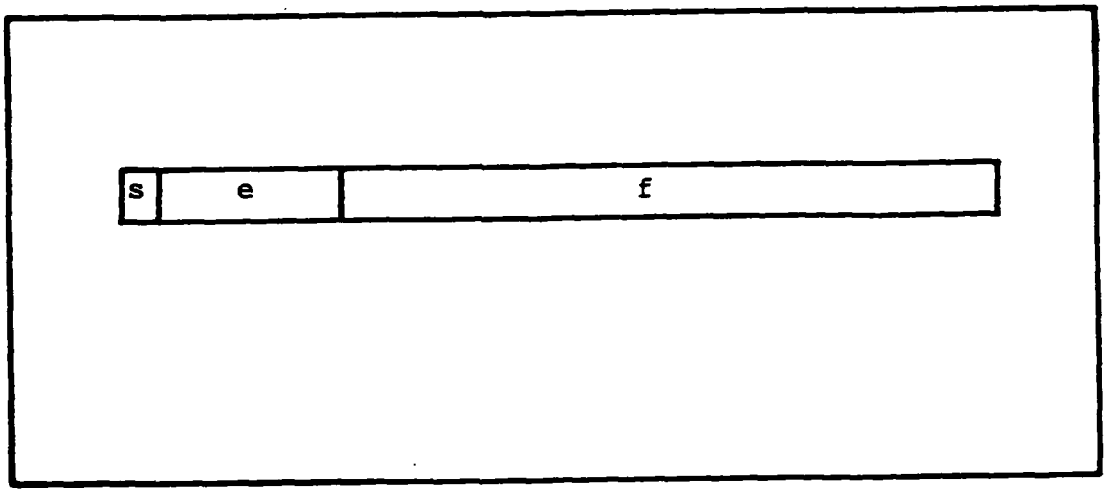
1) An area of memory must be reserved for the Floating-Point Record (FPR).
(2) The names of the FPAL procedures must be declared to be "external".
(3) FPAL procedure references must be imbedded in our source code where appropriate.
(4) The FPAL procedure used by our program must be linked to our object file.
(5) If we want to use the FPAL procedures in our program, that program cannot use symbols that are reserved for FPAL such as symbolic names beginning with a "commercial at" sign, "@" or names whose second character is "Q" or "?".

Floating Point Record (FPR) procedures, arithmetic procedures, error handling procedures, and interface to FPAL are described in Reference 2 in detail.

Data Formats:

FPAL procedures operate on single-precision binary numbers, either 32-bit integer format or in a 32-bit floating-point format.

The integer format recognized by the FPAL is a positive or negative (two's complement) 32-bit binary number The approximate range of this format is from -2.147×10^9 to $+2.147 \times 10^9$. Single-precision formats in the floating-point accumulator and 8080 memory areas are shown in Figure 6.



where s = sign bit

e = exponent

f = fraction or mantissa

Figure 6. SINGLE PRECISION FORMAT FOR FPAL

The three fields within these formats are:

s.....is the sign bit. Sign magnitude representation wehre s = 0 means positive and s = 1 means negative.

e.....is exponent bits. The exponent is offset by (2^7-1). All zeros and all ones in the exponent field are currently reserved for the floating point zero and the invalid numbers described above.

f.....is fraction bits. When the exponent is nonzero, a 1 bit is assumed at the left of the fraction; the binary point is between the assumed bit and the explicit fraction bit.

The number base for the FPAL is binary. The value of a given binary representation can be formulated as

$$(-1)^s \; 2^{e-(2^7-1)} . (1. + .f)$$

where $e \neq 0$ and $e \neq FF$.

When overflow or underflow occurs during FPAL operations, the correct fraction results but the exponent is "wrapped around". A "wrapped around" exponent is defined to be e_w, where the true (offset) exponent, e_t, can be derived from e_w by considering an expanded range of exponents and

$$\text{on overflow} \qquad e_t = e_w - (3.2^6 - 2)$$

$$\text{on underflow} \qquad e_t = e_w - (3.2^6 - 2) \; .$$

The floating point arithmetic library (FPAL) complete functions and execution times are given in Table 2.

FPAL Procedure	Bytes	Typical Execution Time
Float.-Point Add. (FADD)	463	700
Float.-Point Subtract (FSUB)	463	700
Float.-Point Multiply (FMUL)	404	1500
Float.-Point Divide (FDIV)	342	3600
Conversion from decimal float.-point number to binary (FQFD2B)	725	not given
Conversion from binary float.-point number to decimal (FQFB2D)	1585	not given
Float-to-Fixed Conversion (FIXSD)	178	not given
Fixed-to-Float Conversion (FLTDS)	139	not given
Floating Point Compare (FCMPR)	159	300
Floating Pint Test (FZTST)	56	not given
Floating Point Absolute (FABS)	36	not given
Floating Point Negative (FNEG)	43	not given
Note: All time values are specified in microseconds.		

Table 2. 8080/8085 FPAL FUNCTIONS AND EXECUTION TIMES

c. 24-Bit Special Floating-Point Arithmetic Software Subroutines

This software subroutine has been written in Reference 11 to perform the floating-point arithmetic. But the format that has been used is not our concern and not suitable to our standards. The only reason it was mentioned here is to show how slow execution times it has even with 16-bit mantissa. Its functions and execution times are given in Table 3.

d. The SBC 310 High-Speed Mathematics Unit

The SBC 310 High-Speed Mathematics Unit is a member of a complete line of Intel SBC 80 system expansion modules. In performing high-speed mathematic functions, the Math Unit acts as an intelligent processor slaved to one or more SBC 80 computer masters. The math unit performs its repertoire of 14 arithmetic functions an order of magnitude faster than is possible with software routines.

Its functions and execution times are shown in Table 4 and a detailed description of programming information and principles of operation are explained in Appendix A because it was selected to perform this thesis.

Four different systems and their features have been explained individually as they're chosen to be compared in the beginning of this section. The comparison among the SBC 310, High-Speed Math Unit, the AM9511 Arithmetic Processing Unit, the Floating-Point Arithmetic

OPERATION NAME	MAX EXECUTION TIME
24-Bit Floating-Point Addition (ADD)	1300
24-Bit Floating-Point Subtraction (SUB)	1400
24-Bit Floating-Point Multiplication (MULT)	3400
24-Bit Floating-Point Division (DIV)	4100
24-Bit Floating-Point Comparison (COMPARE)	800
24-Bit Floating-Point Squareroot (SQRT)	17,000
24-Bit Floating-Point Cosine (TRIG 1)	18,000
24-Bit Floating-Point Sine (TRIG 2)	18,000
24-Bit Floating-Point Cos and Sine (TRIG 3)	30,000
24-Bit Floating-Point Arctangent (TRIG 4)	19,000
Note: All time values are specified in microseconds.	

Table 3. 24-BIT SPECIAL FLOATING-POINT ARITHMETIC OPERATIONS NAME AND EXECUTION TIMES

OPERATION NAME	OPERATION CODE	TYPICAL EXECUTION TIME	MAXIMUM EXECUTION TIME
Fixed-Point Multiply (MUL)	0	15	20
Fixed-Point Divide (DIV)	1	26	30
Extended Fixed Point Div. (EDIV)	E	84	100
Float.-Point Multiply (FMUL)	2	84	100
Float.-Point Divide (FDIV)	3	92	110
Float.-Point Add (FADD)	4	33	75
Float.-Point Subtract (FSUB)	5	33	75
Float.-Point Square (FSQR)	6	84	100
Float.-Point Square Root (FSQRT)	7	178	205
Fixed-to-Float. Conversion (FLTDS)	8	72	100
Float.-to-Fixed Conversion (FIXSD)	9	42	85
Float. Point Compare (FCMPR)	A	7	7
Float. Point Test (FZTST)	B	7	7
Exchange (EXCH)	F	4	4

Note: All time values are specified in microseconds.

Listed times do not include time to pass arguments to the MATH UNIT and to read results upon completion; this is typically 20 microseconds.

Table 4. SBC-310 HIGH SPEED MATH UNIT FUNCTIONS AND EXECUTION TIMES

Library (FPAL), and 24-bit Special Floating-Point Software execution times of their operations are shown in Table 5.

The AM9511 APU has the fastest and SBC-310 High Speed Math Unit has the second fastest execution times. Also, if we compare the cost of the units, the AM9511 is only $195 in the market and it is cheaper than SBC-310 Math Unit. But, when the purpose of this thesis concern, as a Kalman Filter, and immediate availability at the Naval Postgraduate School, the SBC 310 High-Speed Math Unit was chosen to perform the arithmetics of the Surface/ Subsurface Bearing-Only Tracking Radar algorithms.

C. PHILOSOPHY OF SOFTWARE DESIGN

The system was designed with the following objectives:

1) Human engineered user interface.
2) Capability of solving maneuvering board problems using Kalman Filter algorithms.
3) Capability to display the result and the error performance of the assumed problem definition.

In order to avhieve these objectives, the following modules were developed:

a) "INT$DATA": This module contains range and bearing information of selected contacts and their observation time intervals for each scan.

b. "MAIN$PROG": This module is used to perform the implementation of Kalman filter algorithms, results and error calculations.

SYSTEM NAMES / OPERATION NAMES	TYPICAL EXECUTION TIMES			
	SBC 310 High Speed Math Unit	AM9511 Arithmetic Proc. Unit	FDAL (Software)	24-Bit Special F.P. Software
32-bit Fixed Point Operations				
FMUL	-	97	-	-
DIV	84	98	-	-
32-bit Floating Point Operations				
FMUL	84	73	1500	3400
FDIV	92	77	3600	4100
FADD	33	27	700	1300
FSUB	33	35	700	1400
FSQR	84.	-	-	-
FSQRT	178	391	-	17,000
Fixed-to-Float Conversion	72	28	-	-
Float-to-Fixed Conversion	42	8	-	-
FCEMPR	7	-	300	800
SINE	3800	1898	-	18,000
COSINE	3800	1920	-	18,000
ARCTG	5000	2496	-	30,000

Note: All time values are specified in microseconds.

Table 5. COMPARISON OF FOUR DIFFERENT SYSTEMS EXECUTION TIME OF OPERATION

c. "DISPLAY": This is used to display the result and error calculations on the CRT.

d. "FLOATING$POINT": This module is used to perform all the necessary floating-point operations and to calculate the cosine and/or sine of a given angle in radians and the arctangent of the ratio of two input parameters.

The whole program is listed in Appendix E. The formats, units and conversion factors used in the program are shown in Table 6. The Kalman filter will be described in Chapter IV.

	FORMAT		UNITS	
	I/O	INTERNAL	I/O	INTERNAL
TIME	xxxx.xx	F.P.	seconds	same
COURSE	xxxx.xx	F.P.	degrees	radians
SPEED	xxxx.xx	F.P.	knots	n.mile/ sec.
BEARING	xxxx.xx	F.P.	degrees	radians
RANGE	xxxx.xx	F.P.	yards	nautical miles

x..... Numeric Character

F.P..... Floating Point Represen-ation

1 nautical mile ..... 2025.3716 yards

1 degree ..... 0.0174532925 radians

1 knot ..... 1 nautical mile/hour

π 3.141593

Table 6. FORMAT, UNIT AND CONVERSION FACTORS USED

IV. KALMAN FILTER

A two dimensional Kalman tracking filter provides a convenient framework for determining optimal filter parameters for x,y tracking for a two-dimensional radar measuring range R and bearing θ.

A. DISCRETE KALMAN FILTER ALGORITHM

The vehicle dynamics can be represented in matrix form as stated earlier.

$$Z_{n+1} = \Phi Z_n + \Gamma a_n \qquad \text{(Ref. 13)}$$

where Z_n, Φ, Γ and a_n were defined in Chapter II.

The discrete form of a Kalman filter rather than continuous form is appropriate because of surface search radar antenna rotation results in scan while measurements of the contact position. The matrix form of the observation equation can be written as

$$w_n = HZ_n + v_n \qquad \text{(Ref. 13)}$$

where

$$w_n = \begin{matrix} x_m(n) \\ y_m(n) \end{matrix}$$

$x_m(n)$ = measured x coordinate at scan n

$y_m(n)$ = measured y coordinate at scan n

$$H = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{matrix}$$

Z_n = Vehicle state vector at scan n

$$v_n = \begin{matrix} v_x(n) \\ v_y(n) \end{matrix}$$

$v_x(n)$ = random noise on x measurement at scan n, $x_m(n)$

$v_y(n)$ = random noise on y measurement at scan n, $y_m(n)$.

Optimal estimates of the contact state vector at scan n, $\hat{Z}_n$ are given by

$$\hat{Z}_n = \tilde{Z}_n + \hat{K}_n(w_n - H\tilde{Z}_n) \qquad \text{(Reference 13)}$$

where

Z_n is the estimated state vector at scan n,

$$\tilde{Z}_n = \Phi \hat{Z}_{n-1}$$

Φ (see Chapter II, Section D)

$\hat{Z}_{n-1}$ = Optimal estimate of target state vector at scan (n-1)

and

$$\hat{Z}_n = \begin{matrix} \hat{x}_n \\ \dot{\hat{x}}_n \\ \hat{y}_n \\ \dot{\hat{y}}_n \end{matrix} \quad , \quad \tilde{Z}_n = \begin{matrix} \tilde{x}_n \\ \dot{\tilde{x}}_n \\ \tilde{y}_n \\ \dot{\tilde{y}}_n \end{matrix}$$

$\tilde{Z}_n$ is the optimum estimate of the state vector after the measurement w_n is processed

$\hat{Z}_n$ is the optimum estimate of the state vector before the measurement w_n is processed

$$K_n = \tilde{P}_n H^T (H\tilde{P}_r H^T + R_n)^{-1} \qquad \text{(Reference 13)}$$

K_n is the gain matrix at scan n and computed for steady-state as

$$K_n = \begin{matrix} A_{xx} & A_{xy} \\ B_{xx}/T & B_{xy}/T \\ A_{yx} & A_{yy} \\ B_{yx}/T & B_{yy}/T \end{matrix}$$

$\tilde{P}_n$ is the estimated covariance matrix of the estimation errors prior to the processing run.

R_n is the covariance matrix for the observation noise.

$$R_n = \begin{matrix} \sigma_x^2(n) & \sigma_{xy}^2(n) \\ \sigma_{xy}^2(n) & \sigma_y^2(n) \end{matrix}$$

where using the polar coordinate system shown in Figure 7.

$$x_m = R \sin \theta$$

$$y_m = R \cos \theta$$

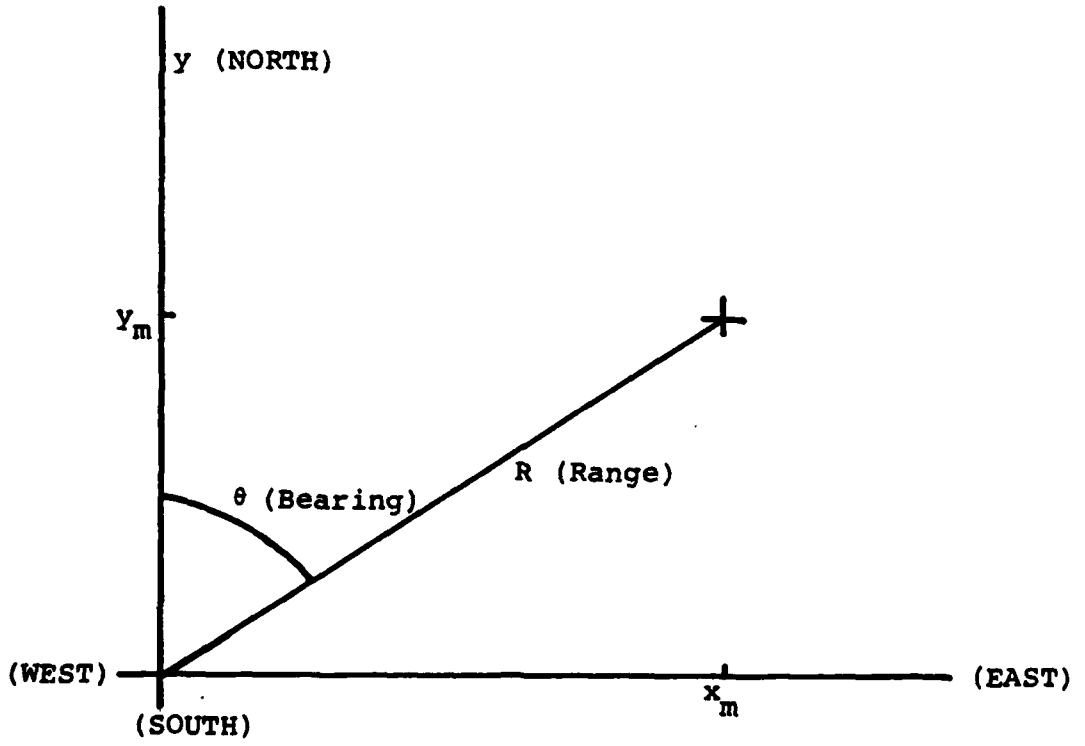


Figure 7. COORDINATE SYSTEM

The elements of the covariance matrix for observation noise are:

$$\sigma_x^2(n) = \sigma_r^2 \sin^2\theta + R^2(n) \cos^2\theta(n)\ \sigma_\theta^2$$

$$\sigma_y^2(n) = \sigma_r^2 \cos^2\theta(n) + R^2(n) \sin^2\theta(n)\ \sigma_\theta^2$$

$$\sigma_{xy}(n) = \frac{1}{2} \sin 2\theta(n) [\sigma_r^2 - R^2(n)\ \sigma_\theta^2]$$

σ_r^2 = variance of the range measurement noise

σ_θ^2 = variance of the bearing measurement.

R(n) and $\theta(n)$ are the vehicle range and bearing at scan n.

The covariance matrix $\tilde{P}_n$ can be computed recursively as

$$\tilde{P}_{n+1} = \Phi \hat{P}_n \Phi^T + \Gamma Q \Gamma^T \qquad \text{(Reference 13)}$$

with $\hat{P}_n = (I - K_n H)\ \tilde{P}_n$

where $\hat{P}_n$ is the covariance matrix of estimation error after processing w_n.

$Q = \sigma_a^2$ and it is the variance of random contact acceleration. The variance of contact acceleration must be equal and independent along x and y axis, and from scan to scan.

This form of the Discrete Kalman filter is valid for our contact maneuver problem. But the solution of the equations in matrix form is unnecessarily complex for the microcomputer. Therefore, an algebraic solution of each element in the matrix form of equations was formulated. These equations are listed in Appendix D.

B. KALMAN FILTER PARAMETERS

The Kalman Filter performance is determined by three parameters:

1) σ_R^2 - Variance of the range measurement error

2) σ_θ^2 - Variance of the bearing measurement error

3) σ_a^2 - Variance of the contact acceleration.

The first two parameters are functions of the selected radar and the ship's gyrocompass. The third parameter is a function of the contact relative motion and the desired smoothness of the output data.

In order to solve the tracking problem and figure out the parameters of the Discrete Kalman Filter, it is necessary to make measurements of

a. Contact Range,

b. Contact Bearing,

c. Own Ship's Course,

d. Own Ship's Speed, and

e. Time of the observation.

It is assumed that all noise is Gaussian and Received signal to noise ratios are large (>>1).

The minimum performance of the selected radar (AN/SPS-10) characteristics are listed in Appendix C and the video signal to noise ratio with single pulse detection is given as

$$\frac{S}{N} = \frac{P_{pk} G^2 \lambda^2 \sigma 10^{-\alpha R/5}}{(4\pi)^3 k T_o F_n B_n L_s R^4} \qquad \text{(Reference 8)}$$

where

P_{pk} = peak transmitted power (Watts)
G = antenna gain
λ = wavelength (m)
σ = radar cross section of contact (m^2)
α = attenuation constant of propagation medium (dB/m)
R = range to contact (m)
k = Boltzman's constant (1.23×10^{-23} joule/deg)
T_o = standard temperature (280°K)
B_n = receiver noise bandwidth (Hz)
L_s = system losses.

The contact range is determined by measuring the time required for a radar pulse to travel to and from the contact.

$$R = \frac{ct}{2}$$

where c = speed of light

t = elapsed time from pulse transmission to pulse detection

R = range to contact.

Threshold detection could be used to indicate the occurrence of the leading edge of the received radar video pulse. Leading edge detection is selected because it eliminates errors resulting from variations in the pulse width. The standard deviation of range error is given below if the rise time of the pulse is limited by receiver bandwidth B.

$$\sigma_R = \frac{e}{2B\left(2\,\frac{S}{N}\right)^{1/2}}$$ (Reference 8)

The contact bearing is determined by measuring the bearing of the radar beam at the time the contact video is detected. Using beam splitting technique to estimate the center bearing of the contact, the standard deviation of bearing error is

$$\sigma_\theta \doteq \frac{1.06\theta_g}{\left[N_g\left(\frac{S}{N}\right)_c\right]^{1/2}}$$

where

σ_θ = standard deviation of bearing error

θ_g = two way beamwidth

$(\frac{S}{N})_c$ = signal to noise ratio at center of beam

N_g = number of pulses emitted as the antenna rotates $2\theta_g$.

The standard deviation of range error and bearing error are functions of the video signal to noise ratio that is given on page 48 for a single pulse detection. When considering the threshold to noise ratio, it is necessary to define both the desired mean time between false alarms and minimum probability of detection of the contact. Probability of false alarms is given by Ref. 8 in terms of the mean time between false alarms and the IF bandwidth of the receiver as shown below.

$$P_{FA} = \frac{1}{T_{FA} B_{IF}} .$$

The mean time between false alarms is a function of the threshold to noise power ratio and the receiver IF bandwidth. And it can be increased if the tracking system is off whenever the contact is not in our selected range. The maximum unambiguous range of AN/SPS-10 Radar is 129.6 nautical miles and for example 50 nm of that is of our

concern. This would disable the measurement circuitry 61.4 percent of the time and it will give us an additional advanatage.

The mean time between false alarms of 35.3 sec (10 times the mean interval) has been selected. The probability of false alarms 5.66×10^{-9} for one contact and 1.93×10^{-8} for 50 nm of our interest range. And this leads to a threshold to noise power ratio of 12.5 dB. Also the reflected radar signals from contact will fluctuate due to cancellation and reinforcement of the waves reflected from various positions of the contact. These fluctuations can be formed as Swerling's Case I and additional 11 to 14 dB will be required to insure a probability of 0.9 to 0.99 resepectively.

The standard deviation for bearing and range can be related to contact range and cross sectional area as given in Ref. 9. Based on a threshold to noise ratio of 12.5 dB and interested range of a 50 nm the standard deviation of the radar range and bearing errors will be 140.2 yards and 0.2062 (0.003599 rad), respectively.

In addition to radar measurement errors also there are ship's gyrocompass and pitometer log errors which come from the own ship's course and speed measurements. And gyrocompass not only determines own ship course but also determines the true bearing of the radar antenna. It is assumed that the standard deviation of own ship's gyrocompass error is 0.1 degrees (0.001745 rad) and standard deviation of own ship's pitometer log error is 0.1 knots.

Therefore, the variance of bearing error is equal to the sum of the radar bearing error ($0.0425\ deg^2$), and the variance of the own ship's gyrocompass error ($0.01\ deg^2$) leads to the total variance of bearing error $0.0525\ deg^2$ or $15.998 \times 10^{-6}\ rad^2$. The variance of range error is $0.0048\ nm^2$.

The variance of target acceleration parameter is the ability of the filter to provide the closest estimate of a contact's position and velocity for a moving contact. It is good to allow the experienced operator to choose the variance of acceleration parameter to be used with each contact.

C. KALMAN FILTER INITIAL CONDITIONS

As it is explained before the Kalman Filter algorithms represent the recursive calculations of the optimal estimates of the covariance of estimation error matrix and the contact state matrix. The initial condition of the contact state matrix is equal to the expected value of the initial state. The expected values of the initial position elements are the first measurements:

$$X_0 = R(0) \sin \theta(0)$$

$$Y_0 = R(0) \cos \theta(0)$$

The expected values of the initial velocity elements are zero.

The initial condition of the covariance of estimation error matrix is equal to the covariance of error of the initial state estimate:

$$\hat{P}(0) = E[\tilde{Z}_0 - \bar{Z}(0)][\tilde{Z}(0) - \bar{Z}(0)]^T$$

$$P(0) = \begin{matrix} E[(\tilde{X}_o - \bar{X}_0)^2 & 0 & 0 & 0 \\ 0 & E[(\dot{\tilde{X}}_0)^2 & 0 & 0 \\ 0 & 0 & E[(\tilde{Y}_0 - \bar{Y}_0)^2] & 0 \\ 0 & 0 & 0 & E[\dot{\tilde{Y}}_0)^2] \end{matrix}$$

The time required for the Kalman Filter to achieve a steady-state condition is very sensitive choosing the closest right values of the nonzero elements of the initial state estimate matrix. They were chosen to be brought the contact on y-axis as an initially by given equations in Reference 13.

V. SYSTEM DESCRIPTION

The description of the system is divided into three major areas: hardware dependencies, system characteristics, and system performance.

A. HARDWARE DEPENDENCIES

Because of hardware equipment that was selected the following hardware dependencies exist in the current implementation of the system:

1. The system utilizes an SBC-310 High-Speed Mathematics Unit to perform floating point arithmetic. Although as discussed before the presence of this math unit could be avoided by replacing its functions with appropriate software routines performing the same operations using the same formats, this is not recommended because of excessive overhead that would result, especially with regard to the execution time. Appendix F explains how the Math Unit was actually implemented.

2. The system depends in three ways on the type of terminal used. The serial asynchronous procedure necessary to communicate between CPU and the terminal, the code needed to control the CRT's functions, and the general features of the DATAMEDIA Elite 2500 Video Terminal notably the programmable roll mode, the setting of privileged fields, the capability of having an addressable cursor, and the possibility of making displayed messages blink.

3. The system occupies approximately 13K bytes of physical memory for code, and approximately 3K bytes to be used for variable data, therefore a configuration of at least 16K bytes of RAM is necessary to execute the system.

B. SYSTEM CHARACTERISTICS

As it was established before, the system was designed to perform basically to solve the maneuvering board problem finding the contact speed and course.

Due to interaction capability between modules as allowed by the language PL/M-80 through the use of attributes PUBLIC and EXTERNAL for the procedures, the following list was written in order to show interactions. This list shows the modules which have procedures called by the listed module.

1. EXTER
2. INT$DATA
3. MAIN$PROG
4. DISPLAY
    a. NUMOUT
    b. DISPLAY$CRT
    c. CRT$READ
    d. CHECKYESNO
    e. DISP$ERROR
5. FLOATING$POINT
    a. MUL
    b. DIV
    c. EDIV
    d. FMUL
    e. FDIV
    f. FADD
    g. FSUB
    h. FSQR
    i. FLTDS
    j. FIXSD

k. FSQRT
l. FCMPR
m. FZTST
n. COS$SIN
o. ARC$TAN

The Module "INT$DATA" is used to implement the radar measurements: range as a yard, bearing as a degree, time of measurements as a second and their interfaces process the system.

The "MAIN$PROG" Module used to perform the Kalman filter algorithms, computation of the filtered target course and speed, display of the result and error performance.

The Module "DISPLAY" includes the display procedures on the CRT

The "FLOATING$POINT" Module is used to perform the floating point arithmetic.

In order to achieve these objectives, a bottom-up implementation philosophy was chosen. Because of the modular design encouraged by PL/M 80 and ISIS-II, it was also possible to map the different levels of design into corresponding modules of software. The basic idea was to encompass all functions corresponding to a level of design into one software module capable of performing all the necessary functions.

C. SYSTEM PERFORMANCE

The performance of the Kalman filter was evaluated for maneuvering contact. The own ship was proceeding on a

steady course (000°) North and at a fixed speed of 30 knots from an initial position at the origin of the cartesian coordinate system as shown in Chapter II, Figure 3. The contact was heading to 060° true (North/East) by 36 knots and its initial position was 009.6° (North/East), 37,800 years from own ship.

The scenario has been made to change the contact course and speed after every 15 sets of measurements. The approximate problem was solved on the maneuvering board and by hand calculator. The first chance has been made for contact course and speed which were 073° (North/South) true and 50 knots, respectively, then second reorientation has been made to 328° (North/West) true course and 30 knots speed.

The accuracy of the computed course and speed of the contact will depend on the tracking system parameters and the maneuverability of the contact. The antenna rotation rate determines the mean time between measurements, which in the case of the AS-1161/SPS Antenna equates to 3.53 seconds. The majority of surface contacts will be capable of turning radii of 100-1000 yards, velocities of 0-50 knots, and acceleration of 0-0.5 yards/sec^2. The following Kalman Filter measurement parameters (σ_θ^2 and σ_r^2) were used as calculated in Chapter IV, Section B.

$$\sigma_r^2 = 0.0048 \text{ nm}^2$$

$$\sigma_\theta^2 = 15.988 \times 10^{-6} \text{ rad}^2 .$$

The Kalman filter initial conditions were the same as those discussed in Chapter IV, Section C. The variance of contact acceleration (σ_a^2) parameter was selected as

$$\sigma_a^2 = 0.0004 \text{ nm}^2/\text{sec}^4$$

Also as is mentioned before, the variance of contact acceleration (σ_a^2) parameter is an independent variable and it is a compromise between the user's desired smoothness of the output data and the ability of the filter to provide the closest estimate of a contact's position and velocity.

This parameter could be selected by enabling one of several read-only memories (ROM) containing the desired values. The particular ROM installed at any one time could be selected from a library of such ROM's by the Commanding Officer, depending on the ship's mission.

The filtered outputs of the contact course and speed according to the selected scenario were plotted on Figure 8 and Fiture 9, respectively. As is noted on those figures, the Kalman Filter achieved a steady state condition after four to seven contact position measurements, corresponding to 14.1 and 24.7 seconds related to the amount of reorientation has been made on contact course and speed. Steady state is defined as the condition where the Kalman Filter Gain Matrix (K_n) would be constant for contact with constant relative position and constant sample intervals.

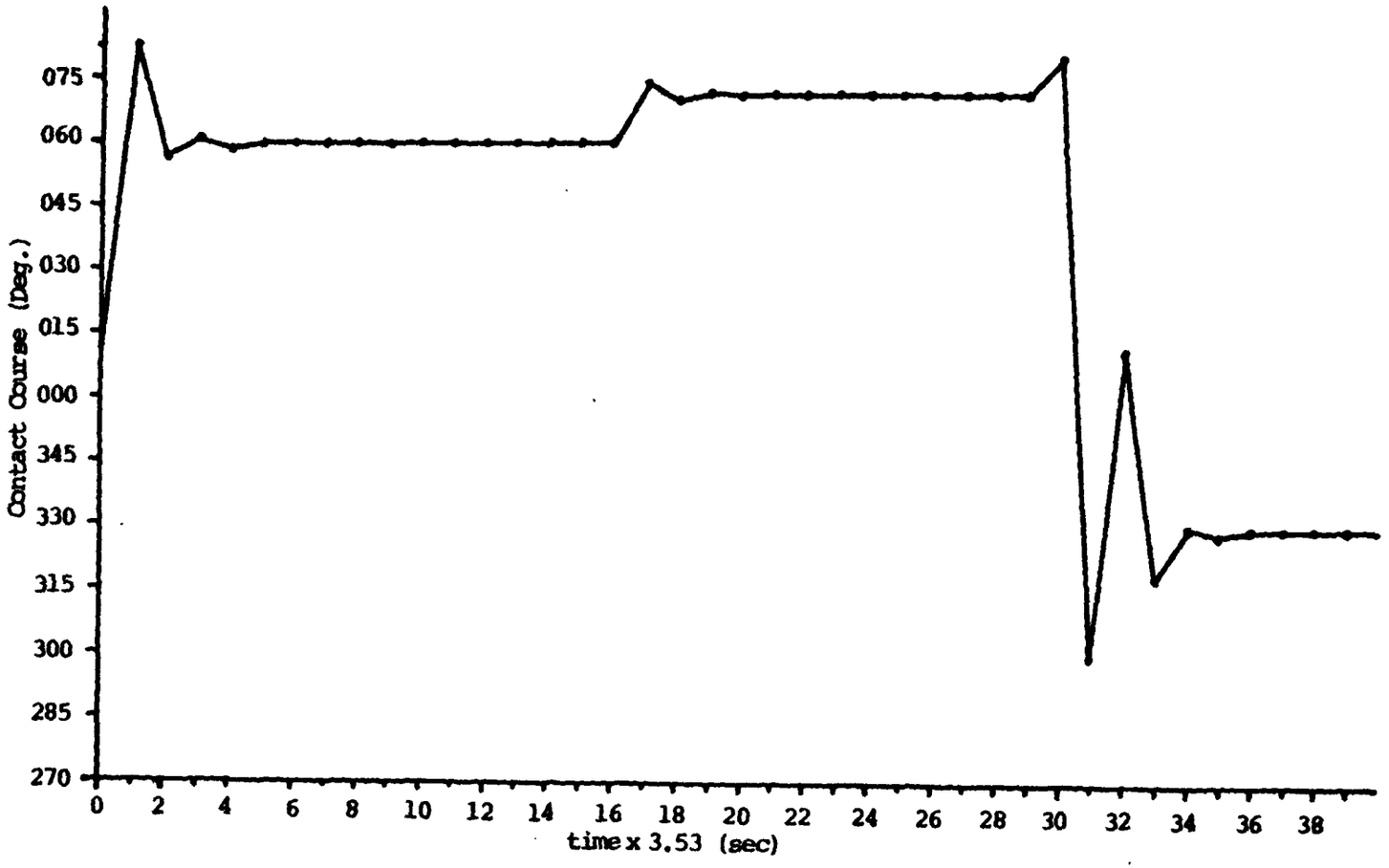


Figure 8. CONTACT COURSE VS. TIME

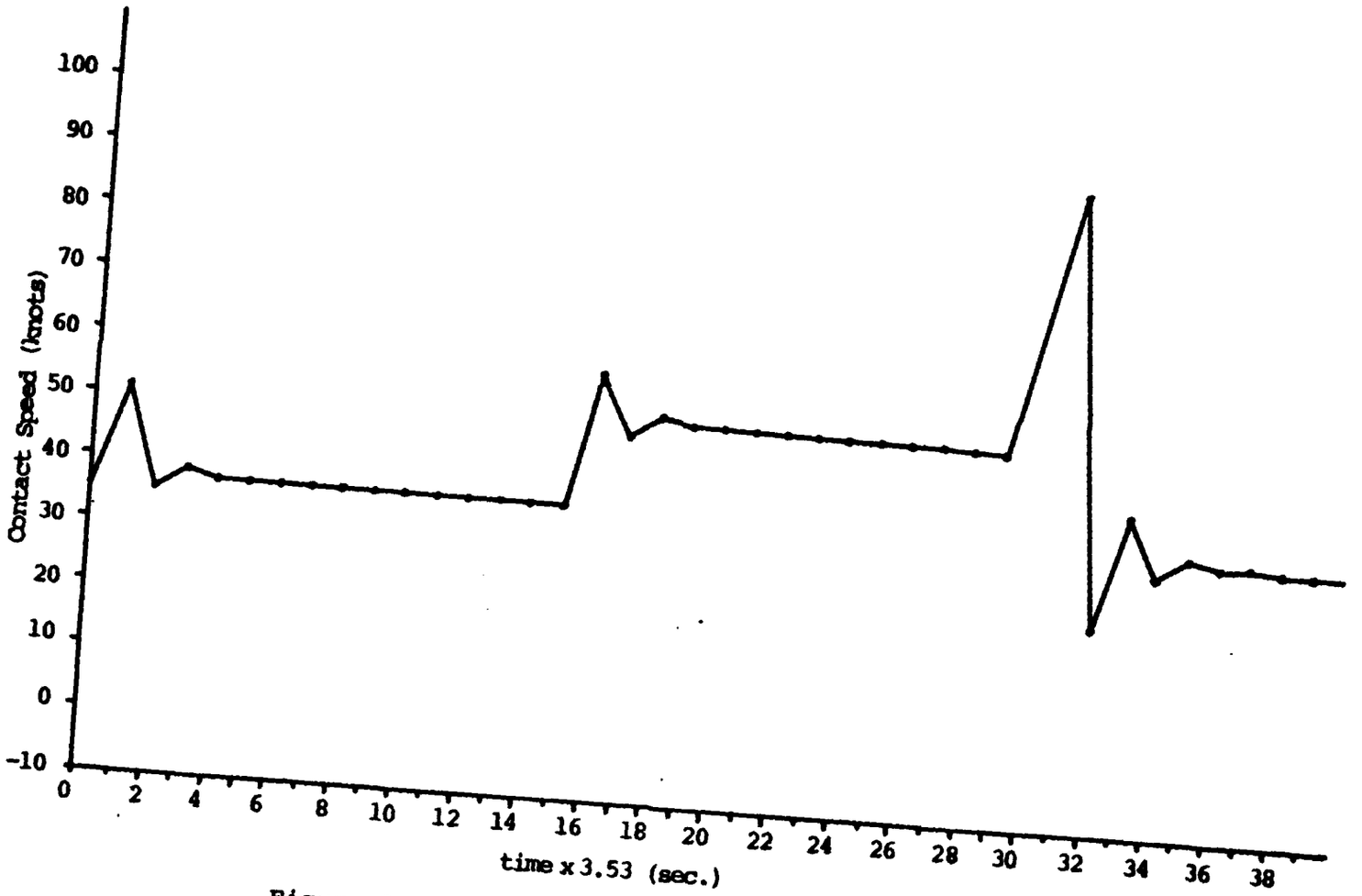


Figure 9. CONTACT SPEED VS. TIME

The bearing, range, course, and speed of contact was computed from the Kalman filter's optimal estimate of contact state and compared with the actual contact bearing, range, course and speed. Figures 10, 11, 12, and 13 show plots of the filtered output of the contact bearing, range, course and speed errors, respectively, as a function of the time interval between measurements.

Several conclusions can be made from these figures. First of all the Kalman filter initial conditions are very much affected on bringing the Kalman filter to steady state condition. Any information about contact speed would cause to shorten the time spent achieving a steady state condition. Also selecting the variance of target acceleration parameter is as important as initiating the filter. The value of the variance acceleration parameter 0.0004 nm^2/sec^2 were chosen after many trials of that selected scenario. It should be noted that the error averages for contact are a function of the elapsed time between maneuvers and the time spent in maneuver.

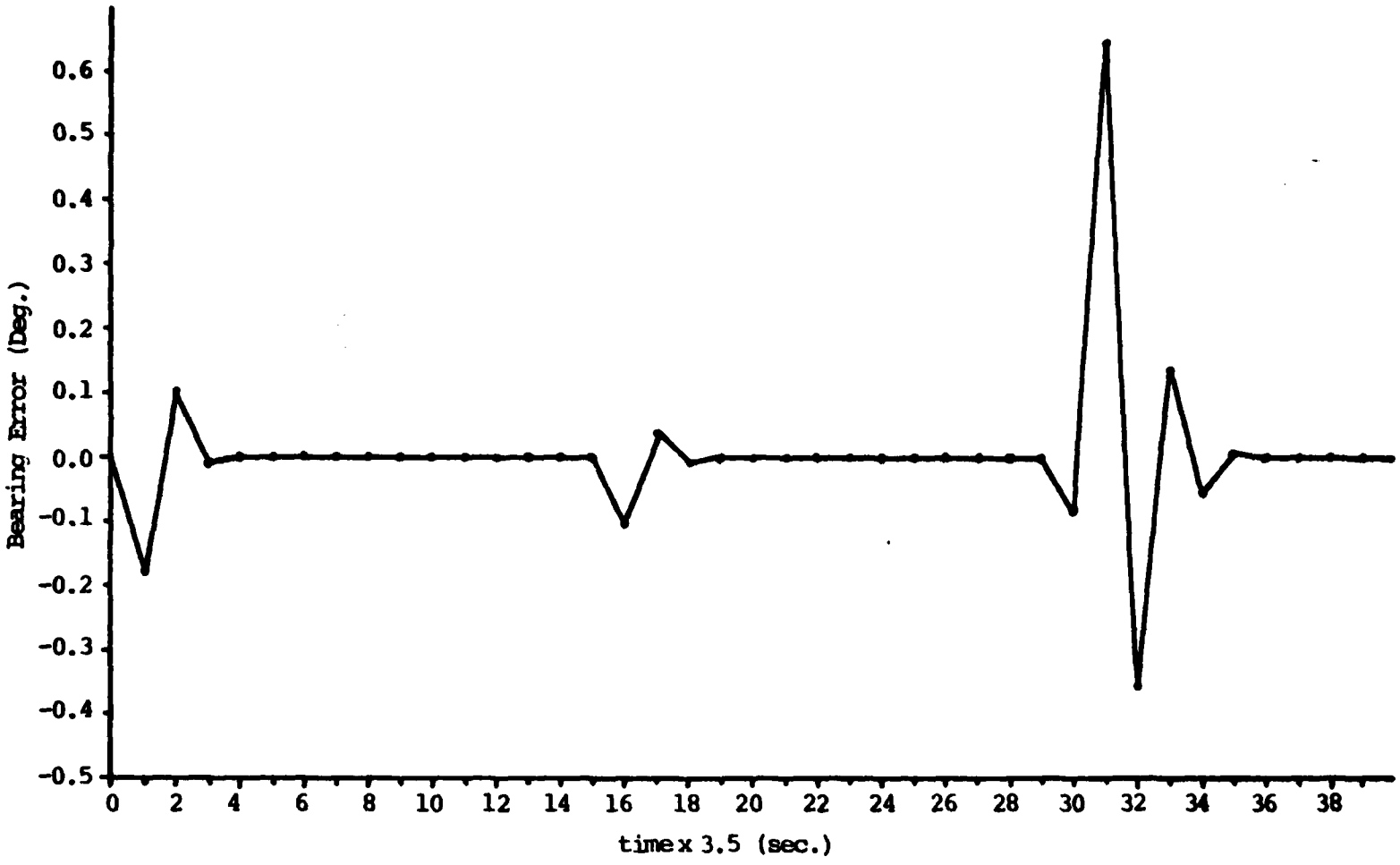


Figure 10. BEARING ERROR VS. TIME

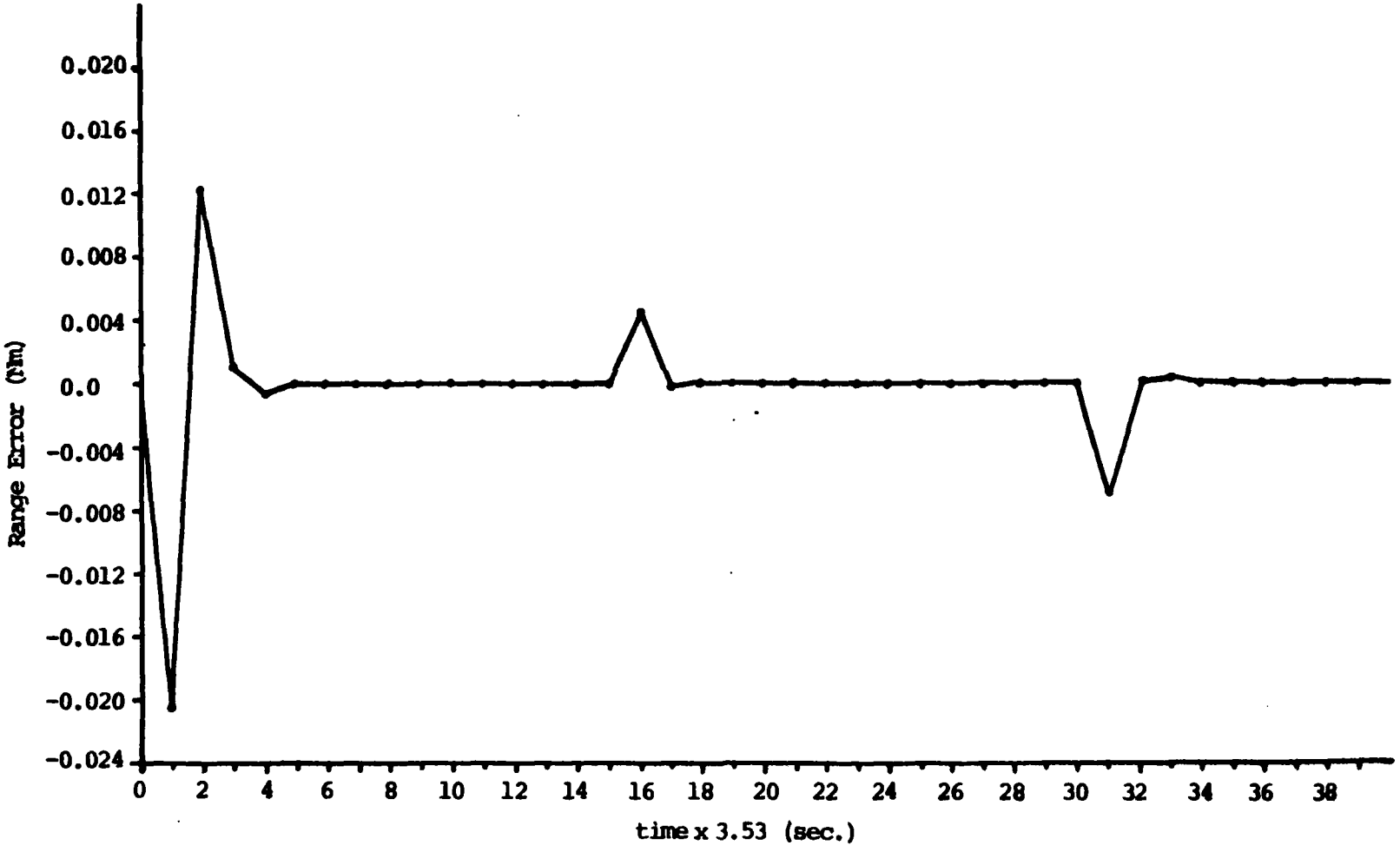


Figure 11. RANGE ERROR VS. TIME

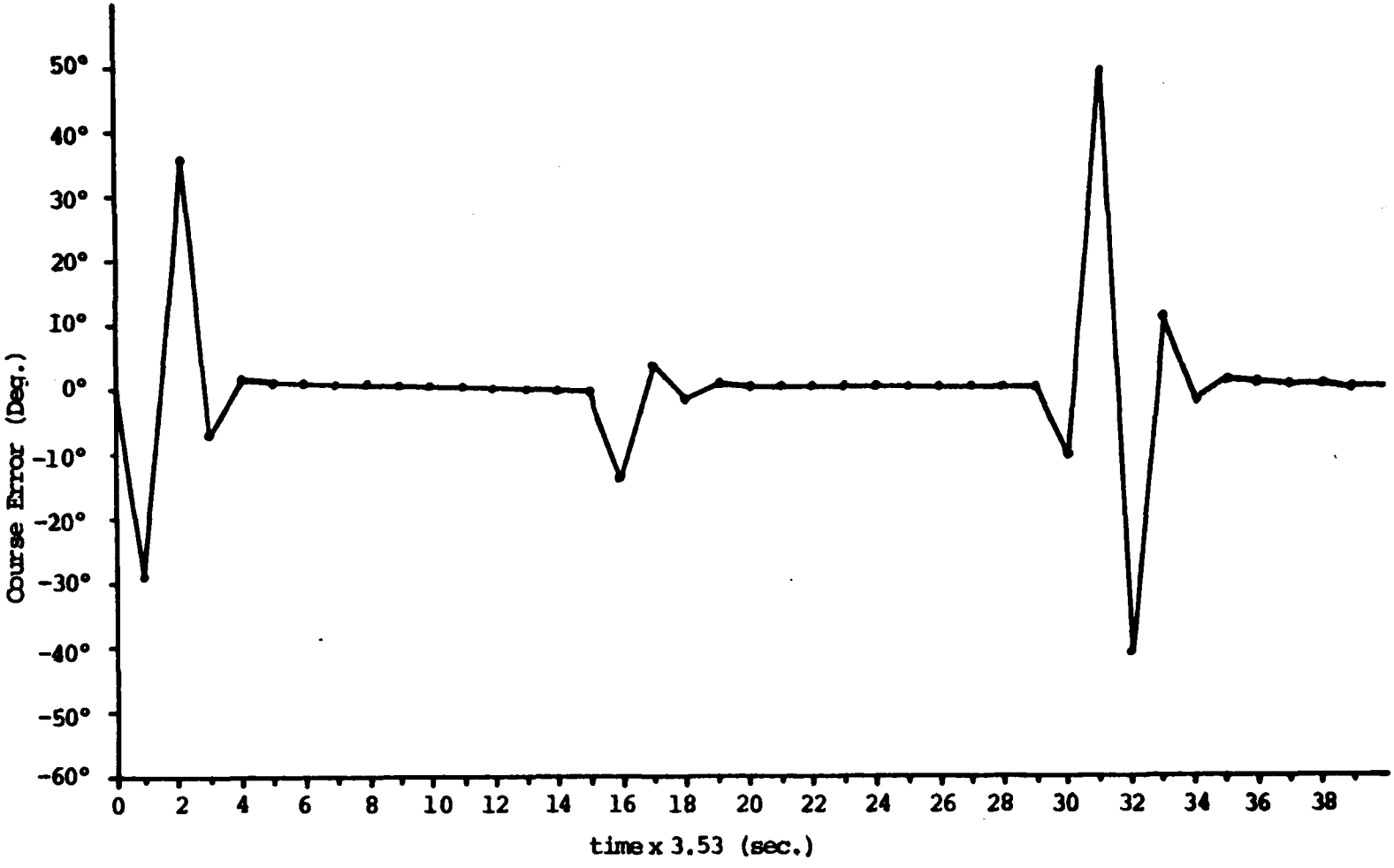


Figure 12. COURSE ERROR VS. TIME

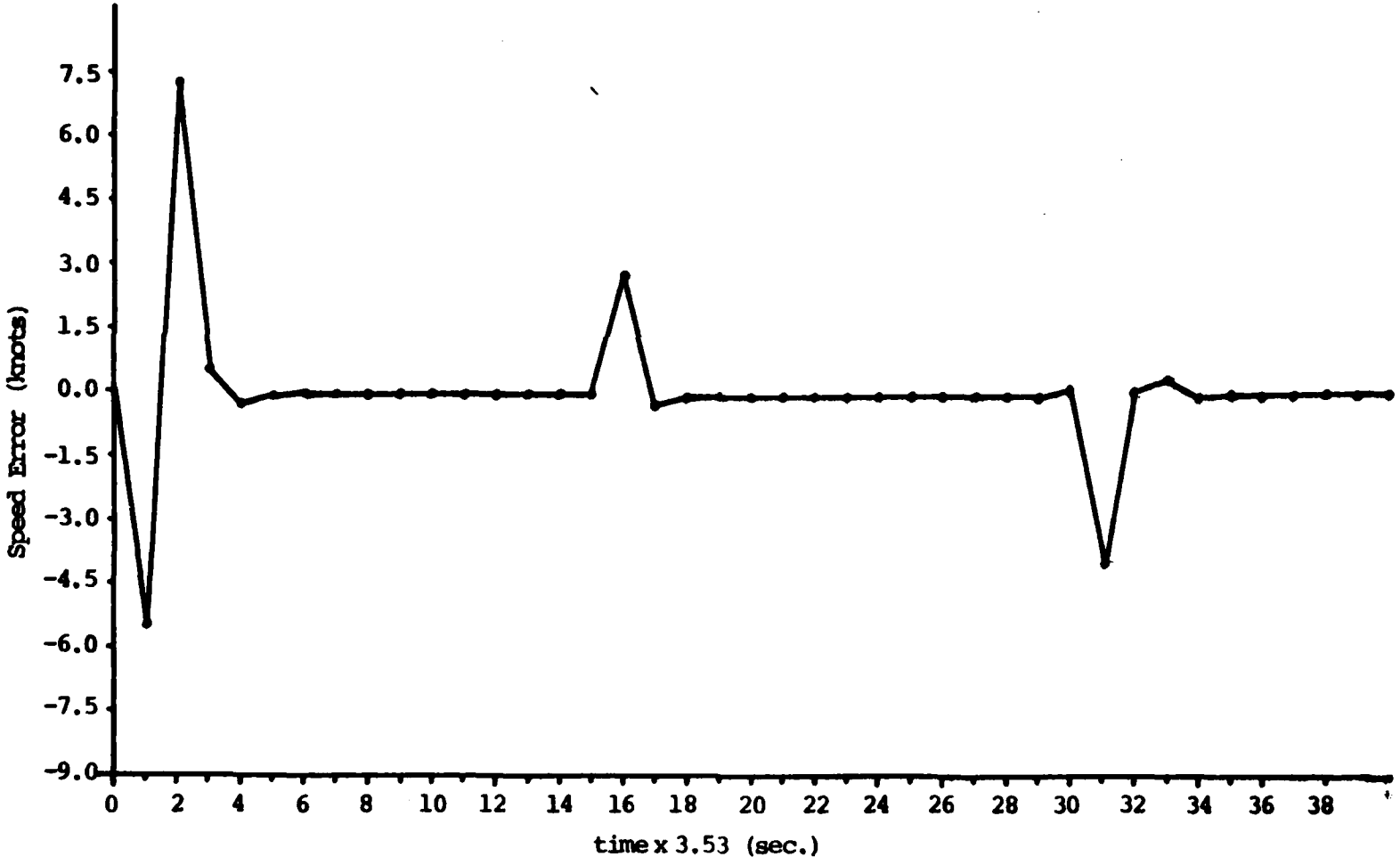


Figure 13. SPEED ERROR VS. TIME

VI. CONCLUSION

A microcomputer-based tracking system has been developed that eliminates an inaccurate and tedious task of manual tracking, for any ship, without the expense of NTOS size equipments, with digital automatic surface/subsurface capability.

It is possible to obtain optimal estimates of contact position and velocity, with the aid of the 4-state Discrete Kalman Filter and appropriate conversions for polar coordinate measurements. Thus the contact course and speed computations were successfully performed by using the Intel intellec MOS microcomputer system, much more accurately than manual tracking techniques.

The SBC-310 High Speed Mathematics Unit has been selected based on a standard floating-point number format with 24 bit mantissa, 7 bit exponent and a sign bit by comparing execution times of procedures to perform sufficient accuracy to reliably compute the Kalman Filter Algorithms and conversion factors.

The total average of the system execution time was approximately 50 msec and with the 3.5 sec sweep of an AS-1161/SPS radar antenna, a single system could continuously provide bearing, range, course and speed data for more than fifty contacts by modifying the software program.

For future work it is possible that the system could compute a contact's closest point of approach (CPA) and could display as (1) CPA Bearing, (2) CPA Range, and (3) Time of CPA by a more sophisticated software study.

The system can be a valuable tool aboard non-NTOS ships as implemented. A microprocessor-based system such as the one described, has the potential to provide any ship with a digital surface/subsurface tracking capability without the expense of NTOS size equipment. It can reduce the manning of underway watches while improving the quality of the tactical information available to the Officer of the Deck.

APPENDIX A

Intel SBC-310 High Speed Mathematical Unit Descriptions

A. GENERAL INFORMATION

The floating-point package developed for this system is based on the SBC-310 High-Speed Mathematics Unit fromIntel Corporation. As described by Reference 1, the SBC 310 Unit is a member of a complete line of the Intel SBC 80 System expansion modules. In performing high-speed mathematical functions, the Math Unit acts as an intelligent processor slaved to one or more SBC 80 Computer Masters. The Mathematic Unit performs its repertoire of 14 arithmetic functions an order of magnitude faster than is possible with software routines.

B. DESCRIPTION OF THE MATH UNIT

The Math Unit is a microprogrammed processor on a single board and is designed to be plugged into a standard SBC 604/614 Modular Backplane and cascade to interface directly with an SBC 80 single board computer or to be used with an Intel Intellec Microcomputer Development System (MDS).

The Math Unit includes the following standart Intel Series 300 shotthy bipolar components: 3001 Microprogram Control Unit (MCU), 3002 Central Processing Element (CPE), 3003 Look-Ahead Carry generator (LCG), and 3604 Electrically Programmable Read Only Memory (PROM). Also included are pipe line register

and bus interface logic. The pipeline register permits the overlapping of microinstruction fetch/execute cycles and the bus interface logic provides compatibility with the Intel Multibus.

Standard Operations include floating point add, subtract, multiply, divide square and square root; fixed point integer multiply, divide, and extended divide; conversion between fixed and floating point representations; and test, compare, and argument exchange operations.

The Math Unit implements unbiased rounding for maximum accuracy. Unbiased rounding is the same as ordinary rounding unless the result is exactly midway between two floating point numbers; in this case ordinary rounding always increases the result, whereas unbiased rounding rounds the result to the nearest even number. When a calculation is performed that results in either an exponent underflow or overflow, the Math Unit provides exponent wraparound to prevent loss of information.

Operation Codes for invoking the arithmetic functions are passed to the Math Unit via I/O Write, Commands, which are also used to initialize the unit with a memory base address. I/O read commands are used to determine the math unit status. Arguments are passed to the Math Unit via Memory Write, Commands and the results are obtained via Memory Read Commands.

The Math Unit which can be operated either in the Interrupt of Polled mode, generates a busy signal during processing

operations and generates either a complete signal or an Error signal after the computation is complete. The information to the host computer which these three signals convey is explained in Ref. (1).

The memory base address and I/O base address are user selectable. The 16-bit memory address is completely under software control and is assigned by the host processor through a sequence of I/O Write Commands, addressed to the Math Unit. The 8 bit I/O base address is selected by a dual inline package (DIP) switch on the board.

All Math Unit operations including arithmetic calculations, data flow between functional elements on the board bus interface, and associated logical tasks, are resident microprogram permanently stored in a set of eight Intel 3604 Erasable Programmable Read Only Memory (EPROM) chips. This memory provides 1,024 micro-instructions of 32 bits each.

C. PREPARTION FOR USE

1. Installation Consideration

The Math Unit is designed for interface with an Intel SBC 80 single board computer based system or an Intel Intellec Microcomputer Development System (MDS).

When installing the SBC 310 in an Intel Intellec MDS, the CPU board needs to be reconfigured in order to generate a Qualified Write Signal; this reconfiguration was obtained by considering the advanced acknowledge (AACK) feature by moving a jumper labelled "advanced write" from a D-C connection

to an E-D connection and by disabling the AACK/LINE (pin 25) on the CPU board as shown in the schematic diagram on page 3-47 of Reference 4.

2. I/O Base Address Switches

The host processor transmits control information and receives status information from the Math Unit by issuing I/O Write and I/O Read Commands, respectively. The I/O address used for these commands is relative to an 8-bit base address that must be a multiple of 8. This base address is assigned by the user by means of an 8-pole dual inline package (DIP) switch assembly. Five of the eight switch poles are connected to the I/O base address detection logic; the other three poles are unused.

The Math Unit had used its DIP switch set to the I/O base address of 10 hexadecimal (only switch pole no. 4 was set on).

3. Programming Information

The I/O base address, which must be assigned by switch selection before the memory base address can be assigned, is normally performed as part of the initial installation procedure. This switch setting allows the user to establish a reference or base to the ports being used in the I/O operations; Table 7 shows the configuration of the I/O addressing as it was set in the system.

The memory base address, which is software controlled, is assigned by a sequence of two I/O Write Commands. The

I/O PORT ADDRESS	OUTPUT	INPUT
P = 010H	OP CODE	R
P+1 = 011H	MEM. LOW	STATUS BYTE
P+2 = 012H	MEM. HIGH	R
P+3 = 013H	R	R
P+4 = 014H	R	R
P+5 = 015H	R	R
P+6 = 016H	R	R
P+7 = 017H	R	FLAG BYTE

P: I/O BASE ADDRESS
R: RESERVED
OP CODE: MATHEMATIC FUNCTION; See Table 4
MEM. LOW: Memory BASE ADDRESS (Lower Byte)
MEM. HIGH: Memory BASE ADDRESS (Upper Byte)

Table 7. I/O ADDRESSING OF SBC-310 HIGH SPEED MATH UNIT

first command is addressed to port (P+1) and loads the low order byte of the memory base address. The second command is addressed to port (P+2) and loads the high order byte of the memory base address. The memory base address must be a multiple of 16; i.e., the lower byte must be in the form XOH (X is any hexadecimal digit) to accommodate the 16 required memory locations used in the arithmetic operations. In the system developed for this thesis the memory base address was set of 0F790H. After both bytes are output, the memory base address (M) is established and need not be reloaded during any subsequent operations. An initialization routine for establishing the memory base address was designed and it can be seen in the "INIT$FP" procedure in the floating point module (see "FLOATING$PINT" module in Appendix F).

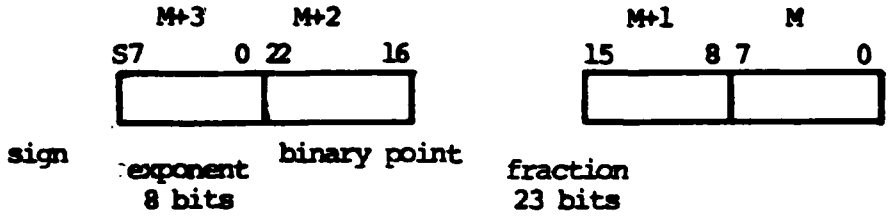
4. Math Unit Functions

The Math Unit performs floating point arithmetic, fixed-point integer arithmetic, compare and test operations, and, float-to-fix and fix-to-float conversions. Operation Codes and execution times for the various functions are listed in Table 4. Arithmetic and conversion formats are shown in Table 8.

Beyond the functions performed by the Math Unit, the floating-point package was designed with two more procedures which compute the cosine and sine of a given angle and the arctangent value of the ratio of given arguments.

SINGLE PRECISION FLOATING POINT

1

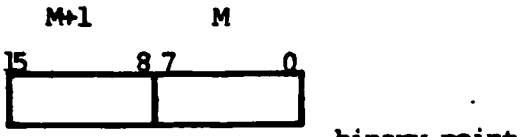


where: M: Memory Base Address
S: "0" = positive; "1" = negative

E7-E0: Biased exponent; Bias = 07FH.
F22-F0: Fraction; F is always normalized

FIXED POINT INTEGER

2



where: M: Memory Base Address

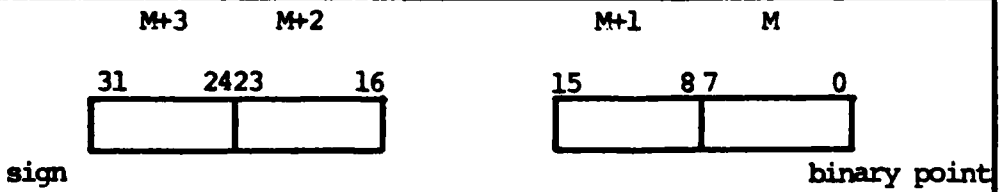
F15-F0: 16-bit integer (unsigned)

2A

where: M: Memory Base Address

F31-F0: 32-bit integer (unsigned)

CONVERSION FUNCTION



where: M: Memory Base Address
S: "0" = positive; "1" = negative

F30-F0: two's complement integer

Table 8. ARITHMETIC AND CONVERSION FORMATS FOR SBC-10 HIGH SPEED MATH UNIT

5. Argument and Result Data Formats

Argument and result data formats and memory locations for the various operations are presented in Table 9. For each argument and result, this table includes a Format number cross-referenced to one of the four formats shown in Table 8. Table 9 also includes the OP CODE for each operation. It is important to note that the result of an operation replaces the first argument in memory, and that the second argument may be destroyed in the course of the computation, these side effects were avoided in the floating-point software design by saving the original values and by allowing the user the possibility of having one of the operands as the result. Error conditions for each operation are described in the next paragraph.

6. Status and Flags

The Math Unit may be operated in the interrupt mode or plled mode.

In the interrupt mode, the Math Unit may be wire-wrapped to initiate an interrupt request under one or both of the following conditions:

a. Operation Complete without an error.

b. Operation Complete with an error.

These "completion" signals may be individually wire-wrapped to separate interrupt lines or both "completion" signals may be wire-wrapped to the same interrupt line (Reference 1).

OP	OP CODE	ARGUMENT FORMAT*	ARGUMENTS**	RESULT FORMAT*	RESULT***
MUL	0	2 2	M,M+1 M+4,M+5	2A	M,M+1,M+2,M+3
DIV	1	1 1	M,M+1 M+4,M+5	2	M,M+1 (remainder in M+4,M+5)
EDIV	E	2A 2	M,M+1,M+2,M+3 M+4,M+5	2	M,M+1,M+2,M+3 (remainder in M+1, M+5,M+6,M+7)
FMUL FDIV FADD FSUB	2 3 4 5	1	M,M+1,M+2,M+3 M+4,M+5,M+6,M+7	1	M,M+1,M+2,M+3
FSQR FSQRT	6 7	1	M,M+1,M+2,M+3	1	M,M+1,M+2,M+3
FLTDS	8	3	M,M+1,M+2,M+3	1	M,M+1,M+2,M+3
FIXSD	9	1	M,M+1,M+2,M+3	3	M,M+1,M+2,M+3
FCMPR	A	1	M,M+1,M+2,M+3 M+4,M+5,M+6,M+7	-	STATUS Byte
FZTST	B	1	M,M+1,M+2,M+3	-	STATUS Byte
EXCH	F	Any	M,M+1,M+2,M+3 M+4,M+5,M+6,M+7	-	Rotates both Arguments

*Refer to appropriate FORMAT No column in Table 8

**Second argument is always the operator, and may be destroyed during operation.

***Results of all operations, except FIXSD are rounded, FIXSD truncates the result.

Table 9. OPERATION ARGUMENT AND RESULT DATA FORMATS FOR SBC-10

In the polled mode, the subroutine designed fo this purpose checks both the status byte and the flag byte (Tables 10 and 11). The polled mode procedure loops on testing the busy bit until the busy bit is clear and then checks the error bits. If an error exists, the error code is input from the Math Unit and a message error is issued. The software has developed for control of the Math Unit if the system employs the polled mode described.

As mentioned before, the condition of the Math Unit is continuously updated and stored. The flag byte shown in Table 10 may be obtained by performing an I/O Read Command to Pt 7 (Table 7). After an operation is completed, the status byte may be obtained by performing an I/O Read Command to P+L (Table 7).

As shown in Table (11), the status byte indicates error conditions where applicable and the results of compare (FCMPR procedure) and Test (FZTST procedure) operations. Each of the six error conditions are defined as follows:

a. Divide by Zero: (001)

This error condition is returned by either "DIV", "EDIV", or "FDIV" procedures to indicate that an attempt was made to divide by zero.

b. Domain Error (010)

This error condition is returned by the "FSQRT" procedure to indicate that the argument was not in the domain of the function; i.e., an attempt was made to take the square root of a negative number.

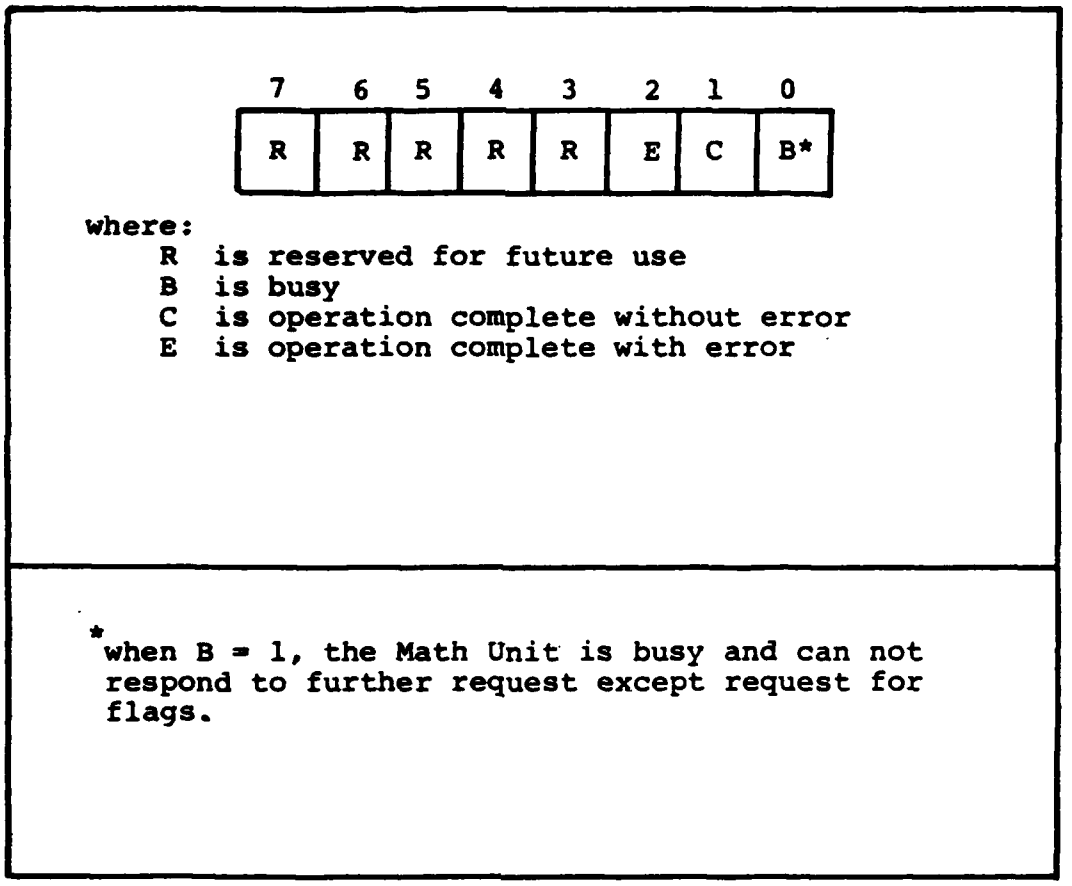
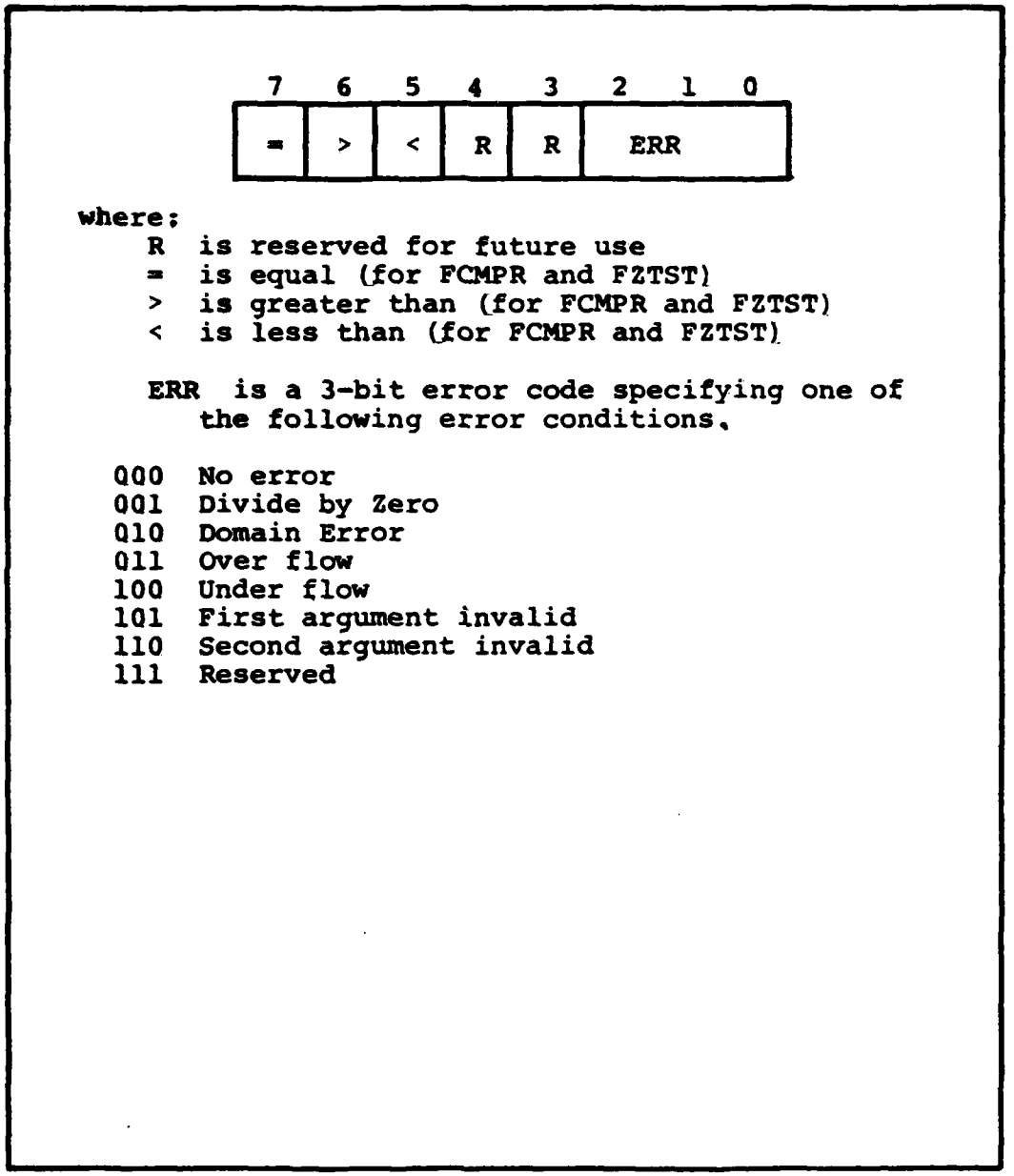


Table 10. FLAG BYTE FORMAT FOR SBC-310



where:

R is reserved for future use
= is equal (for FCMPR and FZTST)
> is greater than (for FCMPR and FZTST)
< is less than (for FCMPR and FZTST)

ERR is a 3-bit error code specifying one of the following error conditions.

000	No error
001	Divide by Zero
010	Domain Error
011	Over flow
100	Under flow
101	First argument invalid
110	Second argument invalid
111	Reserved

Table 11. STATUS BYTE FORMAT FOR SBC-310

c. Overflow (011)

This error condition is returned by the "FADD", "FSUB", "FMIL", "FDIV", "FSOR", and "FIXSD" procedures. In the case of "FIXSD" procedure, this error indicates that the floating-point number is too large to be converted to a 32 bit two's complement signed integer. If an overflow error occurs during "FIXSD", the floating-point argument is left unchaged and may be read from the Math Unit.

In all other cases, this error condition signifies that the exponent of the result is too large to be represented in eight bits. In this case, OBEH is subtracted from the resulting exponent (bringing it back into range for other computations and ensuring a valid result), and the lower eight bits of the exponent are returned in the exponent field of the result.

d. Underflow (100)

This error condition is returned by "FAOO", "FSUB", "FMUL", "FDIV", and "FSQR" procedures to indicate that the exponent of the result is too small to be represented in eight bits. In this case OBEH is added to the resulting exponent bring it back into range for other computations and ensuring a valid result, and the lower eight bits of the exponent are returned in the exponent field of the result.

e. First Argument Invalid (101)

This error condition is returned by the "FADD", "FSUB", "FMUL", "FDIV", "FSQR", "FSQRT", "FIXSD", "FCMPR", and "FZTST" procedures to indicate that the first (or only)

argument for the specified function is invalid. The second argument (if applicable) is not checked if this error is encountered. The invalid argument is left unchanged and may be read from the Math Unit.

f. Second Argument Invalid (110)

This error condition is returned by the "FDD", "FSUB", "FMUL", "FDIV" and "FCMPR" procedures to indicate that the second argument for the specified function is invalid. This error condition occurs only after the first argument is checked and found valid. The invalid argument is left unchanged and may be read from the Math Unit.

Notes:

(1) The floating-point argument may be expressed as:

$$(1) \times (2^{E} - BIAS) \times (1.F)$$

Notice that a "1" is assumed in the highest position of fraction.

(2) There is one unique representation for zero:

S = 0

E7 - E0 = 0

F22 - F0 = 0

(3) The following representations are invalid.

(a) Assures unique representation of zero

$E7 - E0 = 0$, and

$S \neq 0$ or $F22 - F0 = 0$

(b) Reserved for future enhancements

$E7 - E0 = 0FFH$

7. Examples of Floating-Point Number Representations

As shown in Table 8 the representation of a floating-point number has one particularity that needs to be mentioned; i.e., a "1" is always assumed in the highest bit position, and then it yields an effective 24-bit mantissa. For the sake of clarity some examples are given in Table 12.

F.P. Number	M	M+1	M+2	M+3
- INFINITY	OFFH	OFFH	OFFH	OFFH
- 255.0	OOH	OOH	O7FH	OC3H
- 5.0	OOH	OOH	OAOH	OCOH
- 1.0	OOH	OOH	O8OH	O8FH
0.0	OOH	OOH	OOH	OOH
+ 1.0	OOH	OOH	O8OH	O3FH
+ 5.0	OOH	OOH	OAOH	O4OH
+ 255.0	OOH	OOH	O7FH	O43H
+ INFINITY	OFFH	OFFH	OFFH	O7FH
π	ODBH	OOFH	O49H	O4OH
*	OFFH	OFFH	O7FH	O7FH
**	OOH	OOH	O8OH	OOH
1.07×10^9 ***	OFFH	OFFH	O7FH	O4BH

* This is the largest number in the single-precision floating-point format.

** This is the smallest positive number in the single precision floating-point format.

*** This is the largest number that can be converted to floating point without losing accuracy.

Table 12. FLOATING POINT NUMBERS

APPENDIX B

System Start-Up Procedure

Caution: Never turn on or off the diskette drive with a diskette inserted !!!

1. TURN ON MDS SYSTEM. Use the key located at the upper left corner of the front panel and turn it clockwise. The power indicator should light.
2. TURN ON DISKETTE DRIVE. Use power switch located at the front panel. The on indicator should light.
3. TURN ON DATAMEDIA TERMINAL (CRT). Use switch located on right side. The cursor should appear at the screen after a few seconds. Ensure that the lights CD, CTS, ROLL and FULL DUPALLEX are on.
4. PLACE SYSTEM DISKETTE IN DIRVE 0 and the source diskette in drive 1, with the read/write access slot first. Close door of the drives after insertion.
5. BOOTSTRAP THE ISIS-II OPERATING SYSTEM.
   a. Press top of the Intellec BODY Switch.
   b. Press top of the RESET Switch.
   c. Observe that INTERRUPT 2 INDICATOR goes on before proceding.
   d. Press space-bar of Datamedia video terminal keyboard.
   e. Observe that INTERRUPT 2 indicator goes off before proceding.
   f. Press bottom of BOOT Switch.

g. Observe that the following message appears at the Datamedia Video terminal screen:

ISIS-II, V3.4

6. Issue the following Command:

- :F1: FATIH

7. After a few seconds, the Datamedia Video terminal screen will show the first computation of COURSE, BEARING, RANGE, SPEED, THE KALMAN FILTER GAIN MATRIX ELEMENTS, AXX, AXY, AYX, AYY, BXX, BXT, BTX, BYY and the FILTER Error Performance DELTA RANGE, DELTA BEARING, DELTA RANGE DOT, DELTA BEARING DOT and following message appears at the screen:

NEXT ? (Y/N)

8. PRESS THE LETTER 'Y' and see 'YES' message appear on the screen to compute next measurement.

APPENDIX C

AN/SPS-10 Search Radar Characteristics with AS-1161/SPS Radar Antenna

Peak Power,	190 KW
Antenna Gain,	30 dB
Pulse Repetition Rate	625 Hz
Frequency	5825 MHz
Propagation Attenuation	0.6×10^{-5} dB/m
Noise Figure	22.9 dB
Band width	5 MHz
Noise Bandwidth	(1.05 x B) $\simeq$ 5.25 MHz
System tosses	5 dB
Antenna Rotation Rate	17 rpm
Antenna Horizontal Beamwidth	1.9°

APPENDIX D

Algebraic Form of Discrete Kalman Filter

1. COVARIANCE MATRIX FOR THE OBSERVATION NOISE

$$R_n = \begin{matrix} \sigma_x^2(n) & \sigma_{xy}^2(n) \\ \sigma_{xy}^2(n) & \sigma_y^2(n) \end{matrix}$$

$$\sigma_x^2(n) = \sigma_r^2 \cdot \sin^2\theta(n) + r^2(n) \cdot \sigma_\theta^2 \cdot \cos^2\theta(n)$$

$$\sigma_y^2(n) = \sigma_r^2 \cdot \cos^2\theta(n) + r^2(n) \cdot \sigma_\theta^2 \cdot \sin^2\theta(n)$$

$$\sigma_{xy}^2(n) = \frac{1}{2} \cdot \sin 2\theta(n) \cdot [\sigma_r^2 - r^2(n)\sigma_\theta^2]$$

2. THE GAIN MATRIX

$$K_n = \tilde{P}(n) \cdot H^t \cdot (H \cdot \tilde{P}(n) \cdot H^T + R_n)^{-1}$$

$$\Delta = (\tilde{P}_{11}(n) + \sigma_x^2(n))(\tilde{P}_{33}(n) + \sigma_y^2(n)) - (\tilde{P}_{31}(n) + \sigma_{xy}^2(n))(\tilde{P}_{13}(n) + \sigma_{xy}^2(n))$$

$$A_{xx}(n) = [\tilde{P}_{11}(n)(\tilde{P}_{33}(n)+\sigma_y^2(n)) - \tilde{P}_{13}(n)(\tilde{P}_{31}(n)+\sigma_{xy}^2(n))]/\Delta$$

$$A_{xy}(n) = [\tilde{P}_{13}(n)(\tilde{P}_{11}(n) + \sigma_x^2(n)) - \tilde{P}_{11}(n)(P_{13}(n) + \sigma_{xy}^2(n))]/\Delta$$

$$A_{yx}(n) = [\tilde{P}_{31}(n)(\tilde{P}_{33}(n) + \sigma_y^2(n)) - \tilde{P}_{33}(n)(\tilde{P}_{31}(n) + \sigma_{xy}^2(n))]/\Delta$$

$$A_{yy}(n) = [\tilde{P}_{33}(n)(\tilde{P}_{11}(n) + \sigma_x^2(n)) - \tilde{P}_{31}(n)(\tilde{P}_{13}(n) + \sigma_{xy}^2(n))]/\Delta$$

$$B_{xx}(n)/T = [\tilde{P}_{21}(n)(\tilde{P}_{33}(n)+\sigma_y^2(n)) - \tilde{P}_{23}(n)(\tilde{P}_{31}(n)+\sigma_{xy}^2(n))]/\Delta$$

$$B_{xy}(n)/T = [\tilde{P}_{23}(n)(\tilde{P}_{11}(n)+\sigma_x^2(n)) - \tilde{P}_{21}(n)(\tilde{P}_{13}(n)+\sigma_{xy}^2(n))]/\Delta$$

$$B_{yx}(n)/T = [\tilde{P}_{41}(n)(\tilde{P}_{33}(n)+\sigma_y^2(n)) - \tilde{P}_{43}(n)(P_{31}(n)+\sigma_{xy}^2(n)]/\Delta$$

$$B_{yy}(n)/T = [\tilde{P}_{43}(n)(\tilde{P}_{11}(n)+\sigma_x^2(n)) - \tilde{P}_{41}(n)(\tilde{P}_{13}(n)+\sigma_{xy}^2(n))]/\Delta$$

3. THE COVARIANCE MATRIX OF ESTIMATION ERROR AFTER PROCESSING THE OBSERVATION

$$\tilde{P}_n = (I - K_n H)\,\tilde{P}_n$$

$$\hat{P}_{11}(n) = \tilde{P}_{11}(n)(1 - A_{xx}) - \tilde{P}_{31}(n)\,A_{xy}$$

$$\hat{P}_{12}(n) = \tilde{P}_{12}(n)(1 - A_{xx}) - \tilde{P}_{32}(n)\,A_{xy}$$

$$\hat{P}_{13}(n) = \tilde{P}_{13}(n)(1 - A_{xx}) - \tilde{P}_{33}(n)\,A_{xy}$$

$$\hat{P}_{14}(n) = \tilde{P}_{14}(n)(1 - A_{xx}) - \tilde{P}_{34}(n)\,A_{xy}$$

$$\hat{P}_{21}(n) = \tilde{P}_{21}(n) - \tilde{P}_{11}(n)\ B_{xx} - \tilde{P}_{31}(n)\ B_{xy}$$

$$\hat{P}_{22}(n) = \tilde{P}_{22}(n) - \tilde{P}_{12}(n)\ B_{xx} - \tilde{P}_{32}(n)\ B_{xy}$$

$$\hat{P}_{23}(n) = \tilde{P}_{23}(n) - \tilde{P}_{13}(n)\ B_{xx} - \tilde{P}_{-3}(n)\ B_{xy}$$

$$\hat{P}_{24}(n) = \tilde{P}_{24}(n) - \tilde{P}_{14}(n)\ B_{xx} - \tilde{P}_{34}(n)\ B_{xy}$$

$$\hat{P}_{31}(n) = \tilde{P}_{31}(n)(1 - A_{yy}) - \tilde{P}_{11}(n)\ A_{yx}$$

$$\hat{P}_{32}(n) = \tilde{P}_{32}(n)(1 - A_{yy}) - \tilde{P}_{12}(n)\ A_{yx}$$

$$\hat{P}_{33}(n) = \tilde{P}_{33}(n)(1 - A_{yy}) - \tilde{P}_{13}(n)\ A_{yx}$$

$$\hat{P}_{34}(n) = \tilde{P}_{34}(n)(1 - A_{yy}) - \tilde{P}_{14}(n)\ A_{yx}$$

$$\hat{P}_{41}(n) = \tilde{P}_{41}(n) - \tilde{P}_{11}(n)\ B_{yx} - \tilde{P}_{31}(n)\ B_{yy}$$

$$\hat{P}_{42}(n) = \tilde{P}_{42}(n) - \tilde{P}_{12}(n)\ B_{yx} - \tilde{P}_{32}(n)\ B_{yy}$$

$$\hat{P}_{43}(n) = \tilde{P}_{43}(n) - \tilde{P}_{13}(n)\ B_{yx} - \tilde{P}_{33}(n)\ B_{yy}$$

$$\hat{P}_{44}(n) = \tilde{P}_{44}(n) - \tilde{P}_{14}(n)\ B_{yx} \div \tilde{P}_{34}(n)\ B_{yy}$$

4. THE COVARIANCE MATRIX OF THE ESTIMATION ERRORS PRIOR TO PROCESSING THE OBSERVATION

$$\tilde{P}(n+1) = \Phi \cdot \hat{P}_n \cdot \Phi^T + \Gamma \cdot Q \cdot \Gamma^T$$

$$\tilde{P}_{11}(n+1) = \hat{P}_{11}(n) + (\hat{P}_{21}(n)+\hat{P}_{12}(n))T + \hat{P}_{22}(n)T^2 + \sigma_a^2 T^4/4$$

$$\tilde{P}_{12}(n+1) = \hat{P}_{12}(n) + \hat{P}_{22}(n)T + \sigma_a^2 T^3/2$$

$$\tilde{P}_{13}(n+1) = \hat{P}_{13}(n) + (\hat{P}_{23}(n) + \hat{P}_{14}(n))T + \hat{P}_{24}(n)T^2$$

$$\tilde{P}_{14}(n+1) = \hat{P}_{14}(n) + \hat{P}_{24}(n)T$$

$$\tilde{P}_{21}(n+1) = \hat{P}_{21}(n) + \hat{P}_{22}(n)T + \sigma_a^2 T^3/2$$

$$\tilde{P}_{22}(n+1) = \hat{P}_{22}(n) + \sigma_a^2 T^2$$

$$\tilde{P}_{23}(n+1) = \hat{P}_{23}(n) + \hat{P}_{24}(n)T$$

$$\tilde{P}_{24}(n+1) = \hat{P}_{24}(n)$$

$$\tilde{P}_{31}(n+1) = \hat{P}_{31}(n) + (\hat{P}_{41}(n) + \hat{P}_{32}(n))T + \hat{P}_{42}(n)T^2$$

$$\tilde{P}_{32}(n+1) = \hat{P}_{32}(n) + \hat{P}_{42}(n)T$$

$$\tilde{P}_{33}(n+1) = \hat{P}_{33}(n)+(\hat{P}_{43}(n)+\hat{P}_{34}(n))T+\hat{P}_{44}(n)T^2+\sigma_a^2 T^4/4$$

$$\tilde{P}_{34}(n+1) = \hat{P}_{34}(n) + \hat{P}_{44}(n)T + \sigma_a^2 T^3/2$$

$$\tilde{P}_{41}(n+1) = \hat{P}_{41}(n) + \hat{P}_{42}(n)T$$

$$\tilde{P}_{42}(n+1) = \hat{P}_{42}(n)$$

$$\tilde{P}_{43}(n+1) = \hat{P}_{43}(n) + \hat{P}_{44}(n)T + \sigma_a^2 T^3/2$$

$$\tilde{P}_{44}(n+1) = \hat{P}_{44}(n) + \sigma_a^2 T^2$$

APPENDIX E. MAIN PROGRAM LISTING

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE EXTER
OBJECT MODULE PLACED IN :F1:FATIH1.OBJ
COMPILER INVOKED BY:  PLM80 :F1:FATIH1.PLM


   1          EXTER: DO;
                     /***   externals   ***/
   2    1     declare
                        res$table(8) byte external,
                        mebas address external;

   3    1     crt$write:
                        procedure (char) external;
   4    2        declare char byte; end;

   6    1     crt$print$string:
                        procedure (a) external;
   7    2        declare a address; end;

   9    1     send$crlf:
                        procedure external;
  10    2        end;
  11    1     init$fp:
                        procedure external;
  12    2        end;
```

```
13   1      mul:
               procedure (a,b,c) external;
14   2         declare (a,b,c) address; end;

16   1      div:
               procedure (a,b,c,d) external;
17   2         declare (a,b,c,d) address; end;

19   1      ediv:
               procedure (a,b,c,d) external;
20   2         declare (a,b,c,d) address; end;

22   1      fmul:
               procedure (a,b,c) external;
23   2         declare (a,b,c) address; end;

25   1      fdiv:
               procedure (a,b,c) external;
26   2         declare (a,b,c) address; end;

28   1      fadd:
               procedure (a,b,c) external;
29   2         declare (a,b,c) address; end;

31   1      fsub:
               procedure (a,b,c) external;
32   2         declare (a,b,c) address; end;

34   1      fsqr:
               procedure (a,b) external;
```

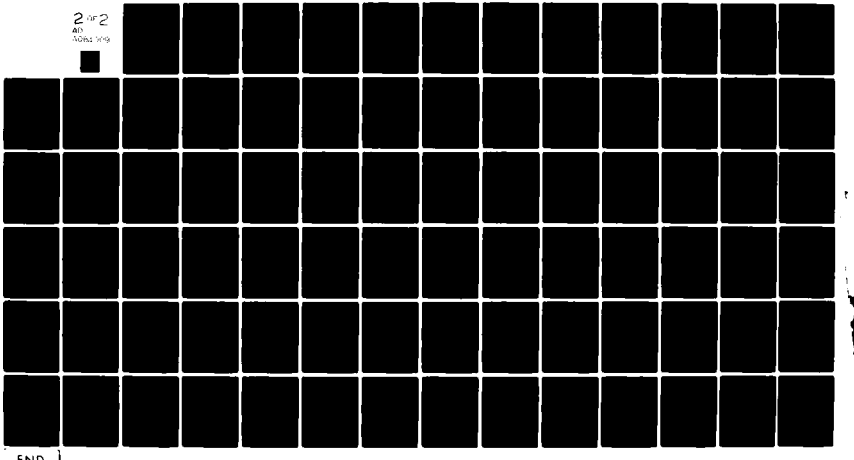
```
 35  2          declare (a,b) address; end;

 37  1      fsort:
            procedure (a,b) external;
 38  2          declare (a,b) address; end;

 40  1      fltds:
            procedure (a,b) external;
 41  2          declare (a,b) address; end;

 43  1      fixsd:
            procedure (a,b) external;
 44  2          declare (a,b) address; end;

 46  1      fcmpr:
            procedure (a,b,c) byte external;
 47  2          declare (a,b,c) address; end;

 49  1      fztst:
            procedure (a,b) byte external;
 50  2          declare (a,b) address; end;

 52  1      exch:
            procedure (a,b) external;
 53  2          declare (a,b) address; end;

 55  1      cos$sin:
            procedure (a,b,c) external;
 56  2          declare (a,b,c) address; end;
```

```
58   1      arc$tan:
               procedure (a,b,c) external;
59   2          declare (a,b,c) address; end;

61   1      numout:
            procedure (a,b) external;
62   2      declare (a,b) address; end;

64   1      display$crt:
            procedure (a) external;
65   2      declare a address; end;

67   1      input$data$interface:
            procedure (a,b,c,d) external;
68   2      declare a byte, (b,c,d) address; end;

70   1      disp$error:
            procedure (a) external;
71   2      declare a address; end;

73   1      end exter;
```

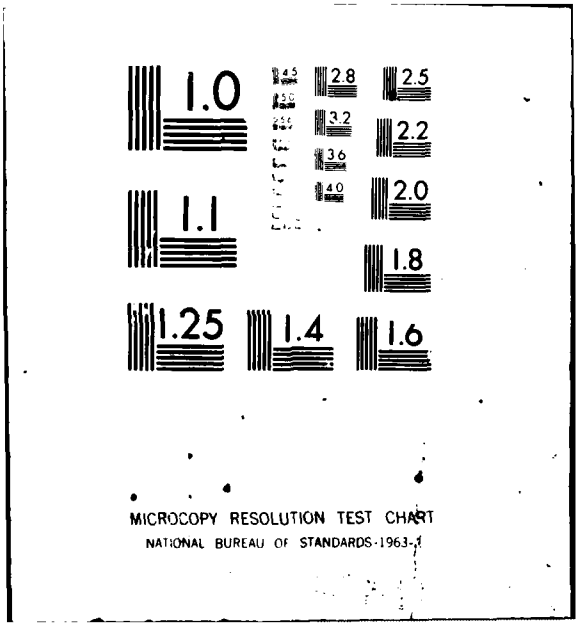
AD-A084 009 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 17/3
ARITHMETIC ARCHITECTURE FOR SURFACE/SUBSURFACE BEARING - ONLY R--ETC(U)
DEC 79 F ERDOGDU

UNCLASSIFIED NL

2 OF 2
AD
A084 009



END
DATE FILMED
6-80
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE INTDATA
OBJECT MODULE PLACED IN :F1:FATIH3.OBJ
COMPILER INVOKED BY: PLM80 :F1:FATIH3.PLM

```
   1          INT$DATA: DO;

   2    1     INPUT$DATA$INTERFACE:PROCEDURE(I,B,C,D) PUBLIC;

   3    2     DECLARE (I,N,N1,N2,N3) BYTE;
   4    2     DECLARE (B,C,D) ADDRESS,
                MRNG BASED B (4) BYTE,
                MBRG BASED C (4) BYTE,
                MTIM BASED D (4) BYTE;

   5    2     DECLARE RNG (*) BYTE DATA (000H,0A8H,013H,047H,
                      000H,094H,013H,047H,
                      000H,080H,013H,047H,
                      000H,06CH,013H,047H,
                      000H,058H,013H,047H,
                      000H,044H,013H,047H,
                      000H,030H,013H,047H,
                      000H,01CH,013H,047H,
                      000H,008H,013H,047H,
                      000H,0F4H,012H,047H,
                      000H,0E0H,012H,047H,
                      000H,0CCH,012H,047H,
                      000H,0B8H,012H,047H,
                      000H,0A4H,012H,047H,
                      000H,090H,012H,047H,
                      000H,07CH,012H,047H,
```

```
000H,072H,012H,047H,
000H,068H,012H,047H,
000H,05FH,012H,047H,
000H,054H,012H,047H,
000H,04AH,012H,047H,
000H,040H,012H,047H,
000H,036H,012H,047H,
000H,02CH,012H,047H,
000H,022H,012H,047H,
000H,018H,012H,047H,
000H,00EH,012H,047H,
000H,004H,012H,047H,
000H,0FAH,011H,047H,
000H,0F0H,011H,047H,
000H,0E6H,011H,047H,

000H,0CDH,011H,047H,
000H,0B4H,011H,047H,
000H,09BH,011H,047H,
000H,082H,011H,047H,
000H,069H,011H,047H,
000H,050H,011H,047H,
000H,037H,011H,047H,
000H,01FH,011H,047H,
```

```
000H,005H,011H,047H,
000H,0ECH,010H,047H,
000H,0D3H,010H,047H,
000H,0BAH,010H,047H,
000H,0A1H,010H,047H,
000H,088H,010H,047H,
000H,06EH,010H,047H,
000H,056H,010H,047H,
000H,024H,010H,047H,
000H,0F2H,00FH,047H,
000H,0C0H,00FH,047H,
000H,08FH,00FH,047H,
000H,05CH,00FH,047H,
000H,02AH,00FH,047H,
000H,0F8H,00EH,047H,
000H,0C6H,00EH,047H,
000H,094H,00EH,047H,
000H,062H,00EH,047H,
000H,030H,00EH,047H,
000H,0FEH,00DH,047H,
000H,0CCH,00DH,047H,
000H,09AH,00DH,047H,
000H,068H,00DH,047H,
000H,036H,00DH,047H);
```

```
6   2       DECLARE BRG (*) BYTE DATA (099H,099H,019H,041H,
                  0CCH,0CCH,01CH,041H,
                  000H,000H,020H,041H,
                  033H,033H,023H,041H,
                  066H,066H,026H,041H,
                  099H,099H,029H,041H,
                  0CCH,0CCH,02CH,041H,
                  000H,000H,030H,041H,
                  033H,033H,033H,041H,
                  066H,066H,036H,041H,
                  099H,099H,039H,041H,
                  0CCH,0CCH,03CH,041H,
                  000H,000H,040H,041H,
                  033H,033H,043H,041H,
                  066H,066H,046H,041H,
                  099H,099H,049H,041H,

                  066H,066H,04EH,041H,
                  033H,033H,053H,041H,
                  000H,000H,058H,041H,
                  0CCH,0CCH,05CH,041H,
                  099H,099H,061H,041H,
                  066H,066H,066H,041H,
                  033H,033H,06BH,041H,
                  000H,000H,070H,041H,
                  0CCH,0CCH,074H,041H,
                  099H,099H,079H,041H,
                  066H,066H,07EH,041H,
                  099H,099H,081H,041H,
                  000H,000H,084H,041H,
```

```
066H,066H,086H,041H,
099H,099H,089H,041H,

000H,000H,088H,041H,
033H,033H,087H,041H,
066H,066H,086H,041H,
099H,099H,085H,041H,
0CCH,0CCH,084H,041H,
000H,000H,084H,041H,
033H,033H,083H,041H,
066H,066H,082H,041H,
099H,099H,081H,041H,
0CCH,0CCH,080H,041H,
000H,000H,080H,041H,
066H,066H,07EH,041H,
0CCH,0CCH,07CH,041H,
033H,033H,07BH,041H,
099H,099H,079H,041H,
```

```
066H,066H,076H,041H,
033H,033H,073H,041H,
000H,000H,070H,041H,
0CCH,0CCH,06CH,041H,
099H,099H,069H,041H,
066H,066H,066H,041H,
033H,033H,063H,041H,
000H,000H,060H,041H,
0CCH,0CCH,05CH,041H,
099H,099H,059H,041H,
066H,066H,056H,041H,
033H,033H,053H,041H,
000H,000H,050H,041H,
0CCH,0CCH,04CH,041H,
099H,099H,049H,041H,
066H,066H,046H,041H,
033H,033H,043H,041H);
```

```
7   2       DECLARE TIM (*) BYTE DATA (000H,000H,060H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
                000H,000H,0E0H,040H,
```

```
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
ØØØH,ØØØH,ØEØH,Ø4ØH,
```

```
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H,
                    000H,000H,0E0H,040H);

 8   2      N = 4 * I;                    N1 = N + 1;
10   2      MRNG(0) = RNG(N);             MRNG(1) = RNG(N1);
12   2      MBRG(0) = BRG(N);             MBRG(1) = BRG(N1);
14   2      MTIM(0) = TIM(N);             MTIM(1) = TIM(N1);

16   2      N2 = N + 2;                   N3 = N + 3;
18   2      MRNG(2) = RNG(N2);            MRNG(3) = RNG(N3);
20   2      MBRG(2) = BRG(N2);            MBRG(3) = BRG(N3);
22   2      MTIM(2) = TIM(N2);            MTIM(3) = TIM(N3);
24   2      END INPUT$DATA$INTERFACE;

25   1      END INT$DATA;
```

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE MAINPROG
OBJECT MODULE PLACED IN :F1:FATIH5.OBJ
COMPILER INVOKED BY: PLM80 :F1:FATIH5.PLM


```
  1             MAIN$PROG: DO;

                /*** EXTERNALS ***/

  2   1         FADD :
                PROCEDURE (A,B,C) EXTERNAL;
  3   2         DECLARE (A,B,C) ADDRESS; END;
  5   1         FSUB :
                PROCEDURE (A,B,C) EXTERNAL;
  6   2         DECLARE (A,B,C) ADDRESS; END;
  8   1         FMUL :
                PROCEDURE (A,B,C) EXTERNAL;
  9   2         DECLARE (A,B,C) ADDRESS; END;
 11   1         FDIV :
                PROCEDURE (A,B,C) EXTERNAL;
 12   2         DECLARE (A,B,C) ADDRESS; END;
```

```
14   1      FSQR :
            PROCEDURE (A,B) EXTERNAL;
15   2      DECLARE (A,B) ADDRESS; END;
17   1      FSQRT :
            PROCEDURE (A,B) EXTERNAL;
18   2      DECLARE (A,B) ADDRESS; END;
20   1      ARC$TAN :
            PROCEDURE (A,B,C) EXTERNAL;
21   2      DECLARE (A,B,C) ADDRESS; END;
23   1      COS$SIN :
            PROCEDURE (A,B,C) EXTERNAL;
24   2      DECLARE (A,B,C) ADDRESS; END;
26   1      SEND$CRLF :
            PROCEDURE EXTERNAL;
27   2      END;

28   1      NUMOUT:
            PROCEDURE (A,B) EXTERNAL;
29   2      DECLARE (A,B) ADDRESS; END;

31   1      DISPLAY$CRT:
            PROCEDURE (A) EXTERNAL;
32   2      DECLARE A ADDRESS; END;

34   1      CPT$PRINT$STRING:
               PROCEDURE (A) EXTERNAL;
35   2         DECLARE A ADDRESS; END;

37   1      CHECK$YES$NO:
            PROCEDURE BYTE EXTERNAL; END;

39   1      INIT$FP:
            PROCEDURE EXTERNAL; END;
```

```
41   1      INPUT$DATA$INTERFACE:
            PROCEDURE (I,B,C,D) EXTERNAL;
42   2      DECLARE I BYTE, (B,C,D) ADDRESS;
43   2      END;

44   1      DISP$ERROR:
            PROCEDURE (A) EXTERNAL;
45   2      DECLARE A ADDRESS; END;
47   1      DECLARE LIT LITERALLY 'LITERALLY',
                   DCL LIT 'DECLARE';

48   1      DCL
               (I,ITER) BYTE,
               (MRNG,MBRG,MTIM) (4) BYTE,
               (XESTA,VXESTA,YESTA,VYESTA) (4) BYTE,
               (XESTB,VXESTB,YESTB,VYESTB) (4) BYTE,
               (SINMBRG,COSMBRG) (4) BYTE,
               (T,TSQ) (4) BYTE,
               (SIGAT2,SIGAT3,SIGAT4) (4) BYTE,

               (PA11,PA12,PA13,PA14,PA21,PA22,PA23,PA24) (4) BYTE,
               (PA31,PA32,PA33,PA34,PA41,PA42,PA43,PA44) (4) BYTE,

               (PB11,PB12,PB13,PB14,PB21,PB22,PB23,PB24) (4) BYTE,
               (PB31,PB32,PB33,PB34,PB41,PB42,PB43,PB44) (4) BYTE,
```

```
                (SIGX,SIGY,SIGXY) (4) BYTE,
                (COS2MBRG,SIN2MBRG,MRNG2) (4) BYTE,
                (DELTA,DELTAX,DELTAY) (4) BYTE,
                (PB11X,PB33Y,PB31XY,PB13XY) (4) BYTE,

                (AXX,AXY,AYX,AYY,BXX,BXY,BYX,BYY) (4) BYTE,
                (ONE$AXX,ONE$AYY) (4) BYTE,

                (XM,YM,XFSTA2,YFSTA2) (4) BYTE,
                (BEARING,RANGE,RANGESQR) (4) BYTE,

                (VXESTA$EWVEL,VYESTA$NSVEL) (4) BYTE,
                (COURSE,VX$EW$SQ,VY$NS$SQ) (4) BYTE,
                (SPEED,TEMP) (4) BYTE;
     49   1     DCL
                (DELTA$BRG,DELTA$RNG,DELY$DOT,DELX$DOT) (4) BYTE,
                (DEL$BRG$DOT,DEL$RNG$DOT) (4) BYTE;

     50   1     DCL
                SIGTH (4) BYTE DATA (0BBH,037H,086H,037H),  /*  0.000016RADSQR */
                SIGR (4)  BYTE DATA (017H,0B7H,0D1H,03CH),  /*  0.0256 NMSQR   */
                SIGA (4) BYTE DATA (017H,0B7H,051H,039H),
                YD$SEC$TO$KNOT (4) BYTE DATA (0DEH,083H,0E3H,03FH), /* 1/.5626 */
                RAD$TO$DEG (4) BYTE DATA (0E0H,02EH,065H,042H),  /* 57.29578 */
                MIL$TO$YD (4) BYTE DATA (0F4H,02BH,0FDH,044H),   /* 2025.3716 */

                NSVEL (4) BYTE DATA (088H,088H,008H,03CH),   /* 30 KNOT */
                EWVEL (4) BYTE DATA (000H,000H,000H,000H),   /* 0' */

                   PI11 (4) BYTE DATA (0E8H,030H,0BBH,03EH),   /* 0.3656 */
                   PI22 (4) BYTE DATA (02CH,059H,03DH,03CH),   /* 0.01156 */
```

```
                PI33 (4) BYTE DATA (039H,097H,0BBH,03EH),   /* 0.37029 */
                PI44 (4) BYTE DATA (02CH,059H,03DH,03CH),   /* 0.01156 */
                PI12 (4) BYTE DATA (000H,098H,07DH,03DH),   /* 0.06191 */
                PI34 (4) BYTE DATA (0CCH,008H,044H,03DH),   /* 0.47860 */
                PI00 (4) BYTE DATA (000H,000H,000H,000H),
                HR$TO$SEC (4) BYTE DATA (000H,000H,061H,045H),  /* 3600 */

                ZERO (4) BYTE DATA (000H,000H,000H,000H),   /* 0.0 */
                ONE (4) BYTE DATA (000H,000H,080H,03FH),    /* 1.0 */
                TWO (4) BYTE DATA (000H,000H,000H,040H),    /* 2.0 */
                THREE (4) BYTE DATA (000H,000H,040H,040H),  /* 3.0 */
                FOUR (4) BYTE DATA (000H,000H,080H,040H),   /* 4.0 */
            FP$5MIL$SEC (4) BYTE DATA (060H,00BH,0B6H,03AH);  /* 5 MILE/SEC */
 51   1     DCL TIME (4) BYTE DATA (000H,000H,060H,040H);     /* 3.5 SEC */

 52   1     DCL MSG$1 (*) BYTE DATA ('COURSE(IN DEGREE) $$'),
                MSG$2 (*) BYTE DATA ('BEARING(IN DEGREE)$$'),
                MSG$3 (*) BYTE DATA ('RANGE (IN MILE)   $$'),
                MSG$4 (*) BYTE DATA ('SPEED (IN KNOT)   $$');
 53   1     DCL MSG$5 (*) BYTE DATA ('NEXT ?     (Y/N)  $$'),
                MSG$6 (*) BYTE DATA ('DELTA BEARING (DEGREE)  $$'),
                MSG$7 (*) BYTE DATA ('DELTA RANGE (MILE)      $$'),
                MSG$8 (*) BYTE DATA ('DELTA BRG. DOT          $$'),
                MSG$9 (*) BYTE DATA ('DELTA RNG. DOT          $$');
```

```
 54   1      DCL MSG$10 (*) BYTE DATA ('
AXX   $$'),
                MSG$11 (*) BYTE DATA ('
AXY   $$'),
                MSG$12 (*) BYTE DATA ('
AYX   $$'),
                MSG$13 (*) BYTE DATA ('
AYY   $$'),
                MSG$14 (*) BYTE DATA ('
BXX   $$'),
                MSG$15 (*) BYTE DATA ('
BXY   $$'),
                MSG$16 (*) BYTE DATA ('
BYX   $$'),
                MSG$17 (*) BYTE DATA ('
BYY   $$');

 55   1      DCL ARRAY BYTE;

 56   1      CALL INIT$FP;


 57   1      I = 0;
 58   1      DO WHILE I < 62;

 59   2      CALL INPUT$DATA$INTERFACE(I,.MRNG,.MBRG,.MTIM);

 60   2      CALL FDIV(.MBRG,.RAD$TO$DEG,.MBRG);

 61   2      CALL COS$SIN (.MBRG,.COSMBRG,.SINMBRG);

 62   2      CALL FDIV(.MRNG,.MIL$TO$YD,.MRNG);
```

```
63   2       IF I = 0 THEN
64   2           do;
65   3         call fadd(.pi11,.zero,.pb11);
```

PL/M-80 COMPILER

PAGE 4

```
66   3         call fadd(.pi22,.zero,.pb22);
67   3         call fadd(.PI33,.zero,.pb33);
68   3         call fadd(.PI44,.zero,.pb44);
69   3         CALL FADD(.PI12,.ZERO,.PB12);
70   3         CALL FADD(.PI00,.ZERO,.PB13);
71   3         CALL FADD(.PI00,.ZERO,.PB14);
72   3         CALL FADD(.PI12,.ZERO,.PB21);
73   3         CALL FADD(.PI00,.ZERO,.PB23);
74   3         CALL FADD(.PI00,.ZERO,.PB24);
75   3         CALL FADD(.PI00,.ZERO,.PB31);
76   3         CALL FADD(.PI00,.ZERO,.PB32);
77   3         CALL FADD(.PI34,.ZERO,.PB34);
78   3         CALL FADD(.PI00,.ZERO,.PB41);
79   3         CALL FADD(.PI00,.ZERO,.PB42);
80   3         CALL FADD(.PI34,.ZERO,.PB43);

81   3         call fadd(.mtim,.zero,.t);
82   3         call fmul(.mrng,.sinmbrg,.xesTB);
83   3         call fmul(.mrng,.cosmbrg,.yesTB);

84   3         call fadd(.FP$5MIL$SEC,.ZERO,.vxesTB);
85   3         call fadd(.FP$5MIL$SEC,.ZERO,.vyesTP);
```

```
 86   3            END;

 87   2               ELSE DO;
 88   3          CALL FADD (.TIME,.TIME,.T);
 89   3          CALL FMUL (.VXESTA,.T,.XESTB);
 90   3          CALL FADD (.XESTB,.XESTA,.XESTB);
 91   3          CALL FADD (.VXESTA,.ZERO,.VXESTB);
 92   3          CALL FMUL (.VYESTA,.T,.YESTB);
 93   3          CALL FADD (.YESTP,.YESTA,.YESTB);
 94   3          CALL FADD (.VYESTA,.ZERO,.VYESTB);
 95   3               END;


 96   2       CALL FSQR (.T,.TSQ);
 97   2       CALL FMUL (.SIGA,.TSQ,.SIGAT2);
 98   2       CALL FMUL (.SIGAT2,.T,.SIGAT3);
 99   2       CALL FMUL (.SIGAT3,.T,.SIGAT4);


100   2       CALL FSQR (.COSMBRG,.COS2MBRG);
101   2       CALL FSQR (.SINMBRG,.SIN2MBRG);
102   2       CALL FSQR (.MRNG,.MRNG2);
```

```
                /*****************************************************
                *
                * COVARIANCE MATRIX FOR THE OBSERVATION NOISE.

                *
                *****************************************************/

                /* 1. SIGX = SIGR * SIN2MBRG + MRNG2 * SIGTH * COS2MBRG  */
  103   2               CALL FMUL (.MRNG2,.SIGTH,.TEMP);
  104   2               CALL FMUL (.TEMP,.COS2MBRG,.TEMP);
  105   2               CALL FMUL (.SIGR,.SIN2MBRG,.SIGX);
  106   2               CALL FADD (.TEMP,.SIGX,.SIGX);


                /* 2. SIGY = SIGR * COS2MBRG + MRNG2 * SIGTH * SIN2MBRG  */
  107   2               CALL FMUL (.MRNG2,.SIGTH,.TEMP);
  108   2               CALL FMUL (.TEMP,.SIN2MBRG,.TEMP);
  109   2               CALL FMUL (.SIGR,.COS2MBRG,.SIGY);
  110   2               CALL FADD (.TEMP,.SIGY,.SIGY);


                /* 3. SIGXY = SINMBRG * COSMBRG (SIGR - MRNG2 * SIGTH)   */
  111   2               CALL FMUL (.MRNG2,.SIGTH,.TEMP);
  112   2               CALL FSUB (.SIGR,.TEMP,.TEMP);
  113   2               CALL FMUL (.SINMBRG,.TEMP,.TEMP);
  114   2               CALL FMUL (.COSMBRG,.TEMP,.SIGXY);
```

```
                /*****************************
                *
                * THE gain MATRIX.
                *
                *****************************/

                /* DELTA = (PB11+SIGX)(PB33+SIGY)-(PB31+SIGXY)(PB13+SIGXY)  */
115   2         CALL FADD (.PB11,.SIGX,.PB11X);
116   2         CALL FADD (.PB33,.SIGY,.PB33Y);
117   2         CALL FADD (.PB31,.SIGXY,.PB31XY);
118   2         CALL FADD (.PB13,.SIGXY,.PB13XY);
                /* DELTA = (PB11X * PB33Y) - (PB31XY * PB13XY)          */

119   2                   CALL FMUL (.PB11X,.PB33Y,.DELTA);
120   2                   CALL FMUL (.PB31XY,.PB13XY,.TEMP);
121   2                   CALL FSUB (.DELTA,.TEMP,.DELTA);


                /* 1. AXX = (PB11 * PB33Y - PB13 * PB31XY) / DELTA       */
122   2                   CALL FMUL (.PB11,.PB33Y,.AXX);
123   2                   CALL FMUL (.PB13,.PB31XY,.TEMP);
124   2                   CALL FSUB (.AXX,.TEMP,.AXX);
125   2                   CALL FDIV (.AXX,.DELTA,.AXX);


                /* 2. AXY = (PB13 *PB11X - PB11 * PB13XY) / DELTA        */
126   2                   CALL FMUL (.PB13,.PB11X,.AXY);
127   2                   CALL FMUL (.PB11,.PB13XY,.TEMP);
128   2                   CALL FSUB (.AXY,.TEMP,.AXY);
129   2                   CALL FDIV (.AXY,.DELTA,.AXY);

                /* 3. BXX/T = (PB21 * PB33Y - PB23 * PB31XY) / DELTA     */
130   2                   CALL FMUL (.PB21,.PB33Y,.BXX);
131   2                   CALL FMUL (.PB23,.PB31XY,.TEMP);
```

```
132   2                 CALL FSUB (.BXX,.TEMP,.BXX);
133   2                 CALL FDIV (.BXX,.DELTA,.BXX);

                  /* 4. BXY/T = (PB23 * PB11X - PB21 * PB13XY) / DELTA     */
134   2                 CALL FMUL (.PP23,.PB11X,.BXY);
135   2                 CALL FMUL (.PB21,.PB13XY,.TEMP);
136   2                 CALL FSUB (.BXY,.TEMP,.BXY);
137   2                 CALL FDIV (.BXY,.DELTA,.BXY);

                  /* 5. AYX = (PB31 * PB33Y - PB33 * PB31XY) / DELTA       */
138   2                 CALL FMUL (.PB31,.PB33Y,.AYX);
139   2                 CALL FMUL (.PB33,.PB31XY,.TEMP);
140   2                 CALL FSUB (.AYX,.TEMP,.AYX);
141   2                 CALL FDIV (.AYX,.DELTA,.AYX);

                  /* 6. AYY = (PB33 * PB11X - PB31 * PB13XY) / DELTA       */
142   2                 CALL FMUL (.PB33,.PB11X,.AYY);
143   2                 CALL FMUL (.PB31,.PB13XY,.TEMP);
144   2                 CALL FSUB (.AYY,.TEMP,.AYY);
145   2                 CALL FDIV (.AYY,.DELTA,.AYY);

                  /* 7. BYX/T = (PB41 * PB33Y - PB43 * PB31XY) / DELTA     */
146   2                 CALL FMUL (.PB41,.PB33Y,.BYX);
147   2                 CALL FMUL (.PB43,.PB31XY,.TEMP);
148   2                 CALL FSUB (.BYX,.TEMP,.BYX);
149   2                 CALL FDIV (.BYX,.DELTA,.BYX);

                  /* 8. BYY/T = (PB43 * PB11X - PB41 * PB13XY) / DELTA     */
150   2                 CALL FMUL (.PB43,.PB11X,.BYY);
151   2                 CALL FMUL (.PB41,.PB13XY,.TEMP);
152   2                 CALL FSUB (.BYY,.TEMP,.BYY);
153   2                 CALL FDIV (.BYY,.DELTA,.BYY);
```

```
                /*****************************************************************
                *
                * THE COVARIANCE MATRIX OF ESTIMATION ERROR AFTER PROCESSING
                * THE OBSERVATION.
                *
                *****************************************************************/

154   2         CALL FSUB (.ONE,.AXX,.ONE$AXX);
155   2         CALL FSUB (.ONE,.AYY,.ONE$AYY);

                /* 1. PA11 = PB11 * ONE$AXX - PB31 * AXY     */
156   2               CALL FMUL (.PB11,.ONE$AXX,.PA11);
157   2               CALL FMUL (.PB31,.AXY,.TEMP);
158   2               CALL FSUB (.PA11,.TEMP,.PA11);

                /* 2. PA12 = PB12 * ONE$AXX - PB32 * AXY     */
159   2               CALL FMUL (.PB12,.ONE$AXX,.PA12);
160   2               CALL FMUL (.PB32,.AXY,.TEMP);
161   2               CALL FSUB (.PA12,.TEMP,.PA12);

                /* 3. PA13 = PB13 * ONE$AXX - PB33 * AXY     */
162   2               CALL FMUL (.PB13,.ONE$AXX,.PA13);
163   2               CALL FMUL (.PB33,.AXY,.TEMP);
```

```
164   2             CALL FSUB (.PA13,.TEMP,.PA13);

                  /* 4. PA14 = PB14 * ONE$AXX - PB34 * AXY      */
165   2             CALL FMUL (.PB14,.ONE$AXX,.PA14);
166   2             CALL FMUL (.PB34,.AXY,.TEMP);
167   2             CALL FSUB (.PA14,.TEMP,.PA14);

                  /* 5. PA21 = PB21 - PB11 * BXX - PB31 * BXY   */
168   2             CALL FMUL (.PB11,.BXX,.PA21);
169   2             CALL FMUL (.PB31,.BXY,.TEMP);
170   2             CALL FADD (.PA21,.TEMP,.PA21);
171   2             CALL FSUB (.PB21,.PA21,.PA21);

                  /* 6. PA22 = PB22 - PB12 * BXX - PB32 * BXY    */
172   2             CALL FMUL (.PB12,.BXX,.PA22);
173   2             CALL FMUL (.PB32,.BXY,.TEMP);
174   2             CALL FADD (.PA22,.TEMP,.PA22);
175   2             CALL FSUB (.PB22,.PA22,.PA22);

                  /* 7. PA23 = PB23 - PB13 * BXX - PB33 * BXY     */
176   2             CALL FMUL (.PB13,.BXX,.PA23);
177   2             CALL FMUL (.PB33,.BXY,.TEMP);
178   2             CALL FADD (.PA23,.TEMP,.PA23);
179   2             CALL FSUB (.PB23,.PA23,.PA23);

                  /* 8. PA24 = PB24 - PB14 * BXX - PB34 * BXY     */
180   2             CALL FMUL (.PB14,.BXX,.PA24);
181   2             CALL FMUL (.PB34,.BXY,.TEMP);
182   2             CALL FADD (.PA24,.TEMP,.PA24);
183   2             CALL FSUB (.PB24,.PA24,.PA24);
```

```
                    /* 9. PA31 = PB31 * ONE$AYY - PB11 * AYX          */
184   2                   CALL FMUL (.PB31,.ONE$AYY,.PA31);
185   2                   CALL FMUL (.PB11,.AYX,.TEMP);
186   2                   CALL FSUB (.PA31,.TEMP,.PA31);

                    /*10. PA32 = PB32 * ONE$AYY - PB12 * AYX          */
187   2                   CALL FMUL (.PB32,.ONE$AYY,.PA32);
188   2                   CALL FMUL (.PB12,.AYX,.TEMP);
189   2                   CALL FSUB (.PA32,.TEMP,.PA32);

                    /*11. PA33 = PB33 * ONE$AYY - PB13 * AYX          */
190   2                   CALL FMUL (.PB33,.ONE$AYY,.PA33);
191   2                   CALL FMUL (.PB13,.AYX,.TEMP);
192   2                   CALL FSUB (.PA33,.TEMP,.PA33);

                    /*12. PA34 = PB34 * ONE$AYY - PB14 * AYX          */
193   2                   CALL FMUL (.PB34,.ONE$AYY,.PA34);
194   2                   CALL FMUL (.PB14,.AYX,.TEMP);
195   2                   CALL FSUB (.PA34,.TEMP,.PA34);

                    /*13. PA41 = PB41 - PB11 * BYX - PB31 * BYY       */
196   2                   CALL FMUL (.PB11,.BYX,.PA41);
197   2                   CALL FMUL (.PB31,.BYY,.TEMP);
198   2                   CALL FADD (.PA41,.TEMP,.PA41);
199   2                   CALL FSUB (.PB41,.PA41,.PA41);
```

```
                        /*14. PA42 = PP42 - PB12 * BYX - PB32 * BYY       */
200    2                        CALL FMUL (.PB12,.BYX,.PA42);
201    2                        CALL FMUL (.PB32,.BYY,.TEMP);
202    2                        CALL FADD (.PA42,.TEMP,.PA42);
203    2                        CALL FSUB (.PB42,.PA42,.PA42);

                        /*15. PA43 = PP43 - PP13 * BYX - PB33 * BYY       */
204    2                        CALL FMUL (.PB13,.BYX,.PA43);
205    2                        CALL FMUL (.PB33,.BYY,.TEMP);
206    2                        CALL FADD (.PA43,.TEMP,.PA43);
207    2                        CALL FSUB (.PB43,.PA43,.PA43);

                        /*16. PA44 = PP44 - PP14 * BYX - PB34 * BYY       */
208    2                        CALL FMUL (.PB14,.BYX,.PA44);
209    2                        CALL FMUL (.PP34,.BYY,.TEMP);
210    2                        CALL FADD (.PA44,.TEMP,.PA44);
211    2                        CALL FSUB (.PB44,.PA44,.PA44);
```

```
                /*****************************************************************
                *
                * THE COVARIANCE MATRIX OF THE ESTIMATION ERRORS PRIOR TO
                * PROCESSING THE OBSERVATION.
                *
                *****************************************************************/

                /*  1. PB11 = PA11 + (PA12+PA21)T + PA22 * TSQ + SIGAT4 / FOUR  */
212   2                CALL FDIV (.SIGAT4,.FOUR,.PB11);
213   2                CALL FMUL (.PA22,.TSQ,.TEMP);
214   2                CALL FADD (.PB11,.TEMP,.PB11);
215   2                CALL FADD (.PA12,.PA21,.TEMP);
216   2                CALL FMUL (.TEMP,.T,.TEMP);
217   2                CALL FADD (.PB11,.TEMP,.PB11);
218   2                CALL FADD (.PB11,.PA11,.PB11);

                /*  2. PB12 = PA12 + PA22 * T + SIGAT3 / TWO    */
219   2                CALL FDIV (.SIGAT3,.TWO,.PB12);
220   2                CALL FMUL (.PA22,.T,.TEMP);
221   2                CALL FADD (.PB12,.TEMP,.PB12);
222   2                CALL FADD (.PB12,.PA12,.PB12);

                /*  3. PB13 = PA13 + (PA23+PA14) * T + PA24 * TSQ    */
223   2                CALL FADD (.PA23,.PA14,.TEMP);
224   2                CALL FMUL (.TEMP,.T,.PB13);
225   2                CALL FMUL (.PA24,.TSQ,.TEMP);
226   2                CALL FADD (.PB13,.TEMP,.PB13);
227   2                CALL FADD (.PB13,.PA13,.PB13);

                /*  4. PB14 = PA14 + PA24 * T          */
228   2                CALL FMUL (.PA24,.T,.PB14);
229   2                CALL FADD (.PB14,.PA14,.PB14);
```

```
                        /*  5. PR21 = PA21 + PA22 * T + SIGAT3 / TWO    */
230   2                        CALL FDIV (.SIGAT3,.TWO,.PB21);
231   2                        CALL FMUL (.PA22,.T,.TEMP);
232   2                        CALL FADD (.PB21,.TEMP,.PB21);
233   2                        CALL FADD (.PB21,.PA21,.PB21);
```

PL/M-80 COMPILER


```
                        /*  6. PB22 = PA22 + SIGAT2                     */
234   2                        CALL FADD (.PA22,.SIGAT2,.PB22);

                        /*  7. PB23 = PA23 + PA24 * T                   */
235   2                        CALL FMUL (.PA24,.T,.PB23);
236   2                        CALL FADD (.PB23,.PA23,.PB23);

                        /*  8. PB24 = PA24  */
237   2                        CALL FADD (.PA24,.ZERO,.PB24);

                        /*  9. PB31 = PA31 + (PA41+PA32) * T + PA42 * TSQ    */
238   2                        CALL FADD (.PA41,.PA32,.TEMP);
239   2                        CALL FMUL (.TEMP,.T,.PB31);
240   2                        CALL FMUL (.PA42,.TSQ,.TEMP);
241   2                        CALL FADD (.PB31,.TEMP,.PB31);
242   2                        CALL FADD (.PB31,.PA31,.PB31);

                        /* 10. PB32 = PA32 + PA42 * T                   */
243   2                        CALL FMUL (.PA42,.T,.PB32);
244   2                        CALL FADD (.PB32,.PA32,.PB32);
```

```
                    /* 11. PB33 = PA33 + (PA34+PA43)*T + PA44 * TSQ + SIGAT4/FOUR */
245   2                     CALL FDIV (.SIGAT4,.FOUR,.PB33);
246   2                     CALL FMUL (.PA44,.TSQ,.TEMP);
247   2                     CALL FADD (.PB33,.TEMP,.PB33);
248   2                     CALL FADD (.PA34,.PA43,.TEMP);
249   2                     CALL FMUL (.TEMP,.T,.TEMP);
250   2                     CALL FADD (.PB33,.TEMP,.PB33);
251   2                     CALL FADD (.PB33,.PA33,.PB33);

                    /* 12. PB34 = PA34 + PA44 * T + SIGAT3 / TWO    */
252   2                     CALL FDIV (.SIGAT3,.TWO,.PB34);
253   2                     CALL FMUL (.PA44,.T,.TEMP);
254   2                     CALL FADD (.PB34,.TEMP,.PB34);
255   2                     CALL FADD (.PB34,.PA34,.PB34);

                    /* 13. PB41 = PA41 + PA42 * T          */
256   2                     CALL FMUL (.PA42,.T,.PB41);
257   2                     CALL FADD (.PB41,.PA41,.PB41);

                    /* 14. PB42 = PA42    */
258   2                     CALL FADD (.PA42,.ZERO,.PB42);

                    /* 15. PB43 = PA43 + PA44 * T + SIGAT3 / TWO    */
259   2                     CALL FDIV (.SIGAT3,.TWO,.PB43);
260   2                     CALL FMUL (.PA44,.T,.TEMP);
261   2                     CALL FADD (.PB43,.TEMP,.PB43);
262   2                     CALL FADD (.PB43,.PA43,.PB43);

                    /* 16. PB44 = PA44 + SIGAT2            */
263   2                     CALL FADD (.PA44,.SIGAT2,.PB44);
```

```
                /*****************************************************************
                *
                * OPTIMUM ESTIMATE OF THE STATE VECTOR BEFORE THE MEASUREMENT.
                *
                *****************************************************************/

264   2            CALL FMUL (.MRNG,.SINMBRG,.XM);
265   2            CALL FMUL (.MRNG,.COSMBRG,.YM);

266   2            CALL FSUB (.XM,.XESTB,.DELTAX);          /* DELTAX */
267   2            CALL FSUB (.YM,.YESTB,.DELTAY);          /* DELTAY */

                /* XESTA = XESTB + AXX * DELTAX + AXY * DELTAY  */
268   2            CALL FMUL (.AXX,.DELTAX,.XESTA);
269   2            CALL FMUL (.AXY,.DELTAY,.TEMP);
270   2            CALL FADD (.XESTA,.TEMP,.XESTA);
271   2            CALL FADD (.XESTA,.XESTB,.XESTA);

                /* VXESTA = VXESTB + BXX * DELTAX + BXY * DELTAY  */
272   2            CALL FMUL (.BXX,.DELTAX,.VXESTA);
273   2            CALL FMUL (.BXY,.DELTAY,.TEMP);
274   2            CALL FADD (.VXESTA,.TEMP,.VXESTA);
275   2            CALL FADD (.VXESTA,.VXESTB,.VXESTA);

                /* YESTA = YESTB + AYY * DELTAY + AYX * DELTAX     */
276   2            CALL FMUL (.AYY,.DELTAY,.YESTA);
277   2            CALL FMUL (.AYX,.DELTAX,.TEMP);
278   2            CALL FADD (.YESTA,.TEMP,.YESTA);
279   2            CALL FADD (.YESTA,.YESTB,.YESTA);

                /* VYESTA = VYESTB + BYY * DELTAY + BYX * DELTAX   */
280   2            CALL FMUL (.BYY,.DELTAY,.VYESTA);
281   2            CALL FMUL (.BYX,.DELTAX,.TEMP);
282   2            CALL FADD (.VYESTA,.TEMP,.VYESTA);
283   2            CALL FADD (.VYESTA,.VYESTB,.VYESTA);
```

```
                /*************************
                *
                *  RESULT.
                *
                *************************/


284   2         CALL ARC$TAN (.YESTA,.XESTA,.BEARING);
285   2         CALL FMUL (.BEARING,.RAD$TO$DEG,.BEARING);

286   2         CALL FSQR (.XESTA,.XESTA2);
287   2         CALL FSQR (.YESTA,.YESTA2);
288   2         CALL FADD (.XESTA2,.YESTA2,.RANGESQR);
289   2         CALL FSQRT (.RANGESQR,.RANGE);

290   2         CALL FADD (.VXFSTA,.EWVEL,.VXFSTA$EWVEL);
291   2         CALL FADD (.VYESTA,.NSVEL,.VYESTA$NSVEL);
292   2         CALL ARC$TAN (.VYESTA$NSVEL,.VXESTA$EWVEL,.COURSE);
293   2         CALL FMUL (.COURSE,.RAD$TO$DEG,.COURSE);

294   2         CALL FSQR (.VXFSTA$EWVEL,.VX$EW$SQ);
```

```
295   2       CALL FSQR (.VYESTA$NSVEL,.VY$NS$SQ);
296   2       CALL FADD (.VX$EW$SQ,.VY$NS$SQ,.SPEED);
297   2       CALL FSQRT (.SPEED,.SPEED);
298   2       CALL FMUL (.SPEED,.HR$TO$SEC,.SPEED);

299   2        CALL SEND$CRLF;
300   2        CALL SEND$CRLF;
301   2       CALL CRT$PRINT$STRING(.MSG$1);
302   2       CALL DISPLAY$CRT(.COURSE);
303   2        CALL SEND$CRLF;
304   2          CALL CRT$PRINT$STRING (.MSG$10);
305   2          CALL DISP$ERROR (.AXX);
306   2        CALL SEND$CRLF;
307   2       CALL CRT$PRINT$STRING(.MSG$2);
308   2       CALL DISPLAY$CRT(.BEARING);
309   2        CALL SEND$CRLF;
310   2          CALL CRT$PRINT$STRING (.MSG$11);
311   2          CALL DISP$ERROR (.AXY);
312   2        CALL SEND$CRLF;
313   2       CALL CRT$PRINT$STRING(.MSG$3);
314   2       CALL DISPLAY$CRT(.RANGE);
315   2        CALL SEND$CRLF;
316   2          CALL CRT$PRINT$STRING (.MSG$12);
317   2          CALL DISP$ERROR (.AYX);
318   2        CALL SEND$CRLF;
319   2       CALL CRT$PRINT$STRING(.MSG$4);
320   2       CALL DISPLAY$CRT(.SPEED);
321   2        CALL SEND$CRLF;
```

```
322   2          CALL CRT$PRINT$STRING (.MSG$13);
323   2          CALL DISP$ERROR (.AYY);
324   2        CALL SEND$CRLF;
325   2        CALL SEND$CRLF;
326   2        CALL SEND$CRLF;
327   2          CALL CRT$PRINT$STRING (.MSG$14);
328   2          CALL FMUL (.BXX,.T,.BXX);
329   2          CALL DISP$ERROR (.BXX);
330   2        CALL SEND$CRLF;


                /**********************************
                *
                *   ERRORS
                *
                **********************************/

                /* DELTA$BRG = (DELTAY * SINMBRG - DELTAX * COSMBRG) / MRNG  */
331   2             CALL FMUL (.DELTAY,.SINMBRG,.DELTA$BRG);
332   2             CALL FMUL (.DELTAX,.COSMBRG,.TEMP);
333   2             CALL FSUB (.DELTA$BRG,.TEMP,.DELTA$BRG);
334   2             CALL FDIV (.DELTA$BRG,.MRNG,.DELTA$BRG);
335   2             CALL FMUL (.DELTA$BRG,.RAD$TO$DEG,.DELTA$BRG);

                /* DELTA$RNG = DELTAX * SINMBRG + DELTAY * COSMBRG   */
336   2             CALL FMUL (.DELTAX,.SINMBRG,.DELTA$RNG);
337   2             CALL FMUL (.DELTAY,.COSMBRG,.TEMP);
338   2             CALL FADD (.DELTA$RNG,.TEMP,.DELTA$RNG);
```

```
                    /* DEL$BRG$DOT =(DELY$DOT * SINMBRG - DELX$DOT * COSMBRG)/MRNG*/
 339    2              CALL FSUB (.VXESTA,.VXESTB,.DELX$DOT);
 340    2              CALL FSUB (.VYESTA,.VYESTB,.DELY$DOT);

 341    2              CALL FMUL (.DELY$DOT,.SINMBRG,.DEL$BRG$DOT);
 342    2              CALL FMUL (.DELX$DOT,.COSMBRG,.TEMP);
 343    2              CALL FSUB (.DEL$BRG$DOT,.TEMP,.TEMP);
 344    2              CALL FDIV (.TEMP,.MRNG,.DEL$BRG$DOT);
 345    2              CALL FMUL (.DEL$BRG$DOT,.RAD$TO$DEG,.DEL$BRG$DOT);

                    /* DEL$RNG$DOT = DELX$DOT * SINMBRG + DELY$DOT * COSMBRG  */
 346    2              CALL FMUL (.DELX$DOT,.SINMBRG,.DEL$RNG$DOT);
 347    2              CALL FMUL (.DELY$DOT,.COSMBRG,.TEMP);
 348    2              CALL FADD (.DEL$RNG$DOT,.TEMP,.DEL$RNG$DOT);
 349    2              CALL FMUL (.DEL$RNG$DOT,.HR$TO$SEC,.DEL$RNG$DOT);
```

```
350   2          CALL CRT$PRINT$STRING (.MSG$7);
351   2          CALL DISP$ERROR (.DELTA$RNG);
352   2         CALL SEND$CRLF;
353   2          CALL CRT$PRINT$STRING (.MSG$15);
354   2          CALL FMUL (.BXY,.T,.BXY);
355   2          CALL DISP$FRROR (.BXY);
356   2        CALL SEND$CRLF;
357   2          CALL CRT$PRINT$STRING (.MSG$6);
358   2          CALL DISP$ERROR (.DELTA$BRG);
359   2         CALL SEND$CRLF;
360   2          CALL CRT$PRINT$STRING (.MSG$16);
361   2          CALL FMUL (.BYX,.T,.BYX);
362   2          CALL DISP$ERROR (.BYX);
363   2        CALL SEND$CRLF;
364   2          CALL CRT$PRINT$STRING (.MSG$8);
365   2          CALL DISP$ERROR (.DEL$BRG$DOT);
366   2         CALL SEND$CRLF;
367   2          CALL CRT$PRINT$STRING (.MSG$17);
368   2          CALL FMUL (.BYY,.T,.BYY);
369   2          CALL DISP$FRROR (.BYY);
370   2        CALL SEND$CRLF;
371   2          CALL CRT$PRINT$STRING (.MSG$9);
372   2          CALL DISP$FRROR (.DEL$RNG$DOT);
373   2         CALL SEND$CRLF;
374   2         CALL SEND$CRLF;


375   2          CALL CRT$PRINT$STRING (.MSG$5);
376   2          ARRAY = CHECK$YES$NO;

377   2       I = I + 1;
378   2       END;
379   1       END MAIN$PROG;
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE DISPLAY
OBJECT MODULE PLACED IN :F1:FATIH6.OBJ
COMPILER INVOKED BY:  PLM80 :F1:FATIH6.PLM
```

```
   1          DISPLAY: DO;

                 /*** EXTERNALS ***/

   2   1       FIXSD:
                    PROCEDURE (A,B) EXTERNAL;
   3   2           DECLARE (A,B) ADDRESS; END;

   5   1       FMUL:
                    PROCEDURE (A,B,C) EXTERNAL;
   6   2           DECLARE (A,B,C) ADDRESS; END;

   8   1       WRITF:
                    PROCEDURE(AFTN,BUFFER,COUNT,STATUS) EXTERNAL;
   9   2           DECLARE(AFTN,BUFFER,COUNT,STATUS) ADDRESS;
  10   2           END;
  11   1       CRT$PRINT$STRING:
                    PROCEDURE (A) EXTERNAL;
  12   2           DECLARE A ADDRESS; END;

  14   1       DECLARE LIT LITERALLY 'LITERALLY',
                       DCL LIT 'DECLARE';
```

```
                /*************************************************************
                *
                * NUMOUT:
                *   PROCEDURE USED TO CONVERT A NUMBER FROM (INTERNAL) BINARY
                *   FORM TO AN ASCII STRING SUITABLE FOR CRT. IT IS WRITTEN
                *   TO BE CALLED BY DISPLAY$CRT PROCEDURE.
                * PARAMETERS:
                *   VALUE: THE NUMBER WHOSE REPRESENTATION IS DESIRED.
                *   BUFADR:THE ADDRESS OF A BUFFER (TWO DECIMAL POINT AND
                *   THREE INTEGER PART OF IT).
                *
                *************************************************************/
15   1          NUMOUT: PROCEDURE (VALUE,BUFADR) PUBLIC;
16   2             DCL (VALUE,BUFADR) ADDRESS,
                       (I) BYTE,
                       (CHARS BASED BUFADR) (1) BYTE;
17   2             DCL DIGITS (*) BYTE DATA ('0123456789');
18   2             DCL EOL LIT '24H',
                       DOT LIT '2EH',
                       NEGAT LIT '2DH';

19   2          DO;
20   3             CHARS(9),CHARS(8) = EOL; END;
22   2          IF VALUE >= 08CA1H THEN DO;
24   3             VALUE = 0FFFFH - VALUE;
25   3             CHARS(1) = NEGAT; END;
```

```
 27   2       DO I = 2 TO 9;
 28   3          IF I = 4 THEN DO;
 30   4                   CHARS(5) = DOT; END;
 32   3                   ELSE DO;
 33   4          CHARS(9 - I) = DIGITS(VALUE MOD 10);
 34   4          VALUE = VALUE / 10;
 35   4                    END;
 36   3          END;
 37   2          I = 0;
 38   2          DO WHILE CHARS(I) = '0' AND I < 8;
 39   3                   CHARS(I) = ' ';
 40   3                   I = I + 1;
 41   3          END;
 42   2       END NUMOUT;
```

```
                /*************************************************************
                *
                * DISPLAY$CRT :
                *   PROCEDURE USED TO DISPLAY ANY FOUR BYTE VARIABLE IN THE
                *   FORM OF 5 INTEGER AND 2 DECIMAL PART OF IT.
                * PARAMETERS :
                *   A : 4 BYTE VARIABLE WHOSE PRINTED REPRESENTATION IS
                *       DESIRED ON THE CRT.
                *
                *************************************************************/
 43   1       DISPLAY$CRT: PROCEDURE (A) PUBLIC;
 44   2          DCL A ADDRESS;
 45   2          DCL C ADDRESS;
 46   2          DCL T BASED A (4) BYTE;
 47   2          DCL BUFFER (128) BYTE;
 48   2          DCL FP$100 (4) BYTE DATA (000H,000H,0C8H,042H);
 49   2          DCL STATUS ADDRESS;

 50   2          CALL FMUL(.T,.FP$100,.T);
 51   2          CALL FIXSD(.T,.C);
 52   2          CALL NUMOUT(C,.BUFFER);
 53   2          CALL WRITE(0,.BUFFER,9,.STATUS);

 54   2       END DISPLAY$CRT;
```

```
                /*****************************************************************
                *
                * CRT$READ :
                *   PROCEDURE USED TO RECIEVE A CHARECTER FROM THE CRT AND RETURN
                *   ITS VALUE TO THE CALLING MODULE.
                * USAGE : UNTYPED PROCEDURE. RETURNS A BYTE VALUE .(ASCII).
                *
                *****************************************************************/

  55   1       CRT$READ:PROCEDURE BYTE PUBLIC;
  56   2          DCL CHAR BYTE;
  57   2          DCL CRTDATA LIT '0F6H',
                     CRTSTATUS LIT '0F7H',
                     CRT$REC$MASK LIT '06H';
```



```
  58   2          DO WHILE (INPUT(CRTSTATUS) AND CRT$REC$MASK) <> CRT$REC$MASK;
  59   3          END;
  60   2          CHAR = INPUT(CRTDATA);
  61   2          IF CHAR >= 80H THEN CHAR = CHAR XOR 80H;
  63   2          RETURN CHAR;
  64   2          END CRT$READ;
```

```
                /*****************************************************************
                *
                * CHECK$YES$NO :
                *   PROCEDURE USED TO CHECK FOR A VALID YES/NO INPUT FROM THE CRT
                * USAGE : TYPED PROCEDURE. A VALUE OF 1 WILL BE RETURND IF THE
                *   ANSWER IS YES OTHERWISE THE VALUE RETURNED IS 0.
                *
                *****************************************************************/
 65   1         CHECK$YES$NO: PROCEDURE BYTE PUBLIC;
 66   2            DCL CHAR BYTE;
 67   2            CHAR = CRT$READ;
 68   2            DO WHILE (CHAR <> 'Y') AND (CHAR <> 'N')    /* UPPER CASE */
                        AND (CHAR <> 'y') AND (CHAR <> 'n');   /* lower case */
 69   3            CHAR = CRT$READ;
 70   3            END;
 71   2            IF (CHAR = 'Y' ) OR ( CHAR = 'Y' ) THEN DO;
 73   3            CALL CRT$PRINT$STRING (.('YES$$'));
 74   3            RETURN 1; END;
 76   2            ELSE DO;

 77   3            CALL CRT$PRINT$STRING (.('NO $$'));
 78   3            RETURN 0; END;
 80   2         END CHECK$YES$NO;
```

```
 81   1      NUMOUT$ERROR: PROCEDURE (VALUE,BUFADR);
 82   2         DCL (VALUE,BUFADR) ADDRESS,
                    I BYTE,
                    (CHARS BASED BUFADR) (1) BYTE;
 83   2         DCL DIGITS (*) BYTE DATA ('0123456789');
 84   2         DCL EOL LIT '24H',
                    DOT LIT '2EH',
                    EKSI LIT '2DH',
                    APTI LIT '2BH';

 85   2      DO;
 86   3         CHARS(10),CHARS(9) = EOL; END;
 88   2      IF VALUE >= 08CA1H THEN DO;
 90   3         CHARS(7) = EKSI;
 91   3         VALUE = 0FFFFH - VALUE; END;
 93   2      ELSE DO;
 94   3         CHARS(7) = APTI; END;

 96   2      DO I = 4 TO 10;
 97   3         IF I = 8 THEN DO;
 99   4            CHARS(2) = DOT; END;
101   3         ELSE DO;
102   4            CHARS(10 - I) = DIGITS(VALUE MOD 10);
```

```
103   4           VALUE = VALUE / 10;
104   4         END;
105   3       END;

106   2       I = 0;
107   2       DO WHILE CHARS(I) = '0' AND I < 9;
108   3             CHARS(I) = ' ';
109   3             I = I + 1; END;

111   2       END NUMOUT$ERROR;


112   1       DISP$ERROR: PROCEDURE (A) PUBLIC;
113   2         DCL A ADDRESS,
                    C ADDRESS,
                    T BASED A(4) BYTE;
114   2         DCL BUFFER (128) BYTE,
                    STATUS ADDRESS;
115   2         DCL FP$5000 (4) BYTE DATA (000H,040H,09CH,045H);

116   2       CALL FMUL (.T,.FP$5000,.T);
117   2       CALL FIXSD(.T,.C);
118   2       CALL NUMOUT$ERROR (C,.BUFFER);
119   2       CALL WRITE (0,.BUFFER,10,.STATUS);

120   2       END DISP$ERROR;
121   1       END DISPLAY;
```

APPENDIX F. FLOATING-POINT PROGRAM LISTING

```
FLOATING$POINT: DO;

DECLARE LIT LITERALLY 'LITERALLY',
        DCL LIT 'DECLARE';

DCL CR LIT '0DH',                    /* CARRIAGE RETURN.*/
    LF LIT '0AH',                    /* LINE FEED.*/
    EOL LIT '24H',                   /* $$ INDICATES END OF LINE.*/
    BS LIT '08H',                    /* BACKSPACE. */
    SPACE LIT '20H',                 /* SPACE */
    RUB LIT '7FH',                   /* RUB OUT. */
    SUB LIT '1AH',                   /* UP ROW CURSOR. */
    BEL LIT '07H';                   /* PING THE BELL. */
DCL CRTDATA LIT '0F6H',              /* CRT DATA ON PORT 246.*/
    CRTSTATUS LIT '0F7H',            /* CRT STATUS ON PORT 247.*/
    CRT$REC$MASK LIT '06H',          /* MASK FOR CRT: RECEIVE.*/
    CRT$TRT$MASK LIT '05H';          /* MASK FOR CRT: TRANSMIT. */
```

```
DCL
    MEBAS ADDRESS PUBLIC DATA (0F790H) ,
    RES$TABLE (8) BYTE PUBLIC AT (0F790H),  /* MEMORY RESERVED FOR F.P. BOARD. */
    OUT$OP$CODE LIT '010H',   /* BASE OUTPUT PORT FOR F.P. BOARD. */
    IN$STATUS LIT '011H',     /* INPUT PORT FOR STATUS OF F.P. BOARD. */
    IN$FLAG LIT '017H',       /* INPUT PORT FOR FLAG FROM F.P. BOARD. */
    MEM$LOW LIT '011H',       /* OUTPUT PORT FOR LOW PORTION OF MEMORY BASE.*/
    MEM$HIGH LIT '012H',      /* OUTPUT PORT FOR HIGH PORTION OF MEMORY BASE. */
    MUL$CODE LIT '00H',       /* FIXED POINT MULTIPLICATION CODE. */
    DIV$CODE LIT '01H',       /* FIXED POINT DIVISION CODE. */
    FMUL$CODE LIT '02H',      /* F.P. MULTIPLICATION CODE. */
    FDIV$CODE LIT '03H',      /* F.P. DIVISION CODE. */
    FADD$CODE LIT '04H',      /* F.P. ADD CODE. */
    FSUB$CODE LIT '05H',      /* F.P. SUBTRACT CODE. */
    FSQR$CODE LIT '06H',      /* F.P. SQUARE CODE. */
    FSQRT$CODE LIT '07H',     /* F.P. SQUARE ROOT CODE. */
    FLTDS$CODE LIT '08H',     /* FIXED-TO-FLOAT CONVERSION CODE. */
    FIXSD$CODE LIT '09H',     /* FLOAT-TO-FIXED CONVERSION CODE. */
    FCMPR$CODE LIT '0AH',     /* F.P. COMPARE CODE. */
    FZTST$CODE LIT '0BH',     /* F.P. TEST CODE. */
    EXCH$CODE LIT '0FH',      /* EXCHANGE CODE. */
    EDIV$CODE LIT '0EH',      /* EXTENDED FIXED POINT DIVISION CODE. */
    BUSY$MASK LIT '01H',      /* MASK FOR BUSY SIGNAL FROM F.P. BOARD. */
    ERROR$MASK LIT '04H';     /* MASK FOR ERROR CODE FROM F.P. BOARD. */
```

```
/*************************************************************************
*
* CRT$WRITE:
*   PROCEDURE USED TO SEND A CHARACTER TO THE CRT.
*
* PARAMETERS:
*   - CHAR.-CHARACTER BYTE TO BE SENT.
*
*************************************************************************/
CRT$WRITE: PROCEDURE (CHAR) PUBLIC;
   DCL CHAR BYTE;
   DO WHILE (INPUT(CRTSTATUS) AND CRT$TRT$MASK) <> CRT$TRT$MASK;  /*WAIT*/
      END;
   OUTPUT(CRTDATA) = CHAR;
   END CRT$WRITE;


/*************************************************************************
*
* CRT$PRINT$STRING:
*   PROCEDURE USED TO SEND A STRING OF CHARACTERS TO THE CRT.
*
* PARAMETERS:
*   - A.-POINTER TO STRING. $$ WILL INDICATE END OF STRING. MAXIMUM
*        LENGTH: 80 CHARACTERS.
*
*************************************************************************/
CRT$PRINT$STRING: PROCEDURE (A) PUBLIC;
   DCL (POINTER,A) ADDRESS,
       (BUFFER BASED A) (80) BYTE;
   POINTER = 0;
   DO WHILE (BUFFER(POINTER) <> EOL) OR (BUFFER(POINTER + 1) <> EOL);
      CALL CRT$WRITE(BUFFER(POINTER));
      POINTER = POINTER + 1;
      END;
   END CRT$PRINT$STRING;
```

```
/*************************************************************
*
* SEND$CRLF:
*   PROCEDURE USED TO SEND BOTH CR AND LF ASCII CHARACTERS TO CRT.
*
*************************************************************/
SEND$CRLF: PROCEDURE PUBLIC;
   CALL CRT$WRITE(CR);
   CALL CRT$WRITE(LF);
   END SEND$CRLF;
```



```
/*************************************************************
*
* INIT$FP:
*   PROCEDURE USED TO INITIALIZE THE FLOATING POINT MODULE.
*
* USAGE:
*   THIS PROCEDURE SHOULD BE CALLED ONE TIME FROM THE USER'S
*   MAIN PROGRAM BEFORE ATTEMPTING TO USE ANY OF THE ROUTINES PROVI-
*   DED FOR FLOATING POINT OPERATIONS WITH THE FLOATING POINT BOARD.
*
*************************************************************/
INIT$FP: PROCEDURE PUBLIC;
   OUTPUT(MEM$LOW) = LOW(MEBAS);
   OUTPUT(MEM$HIGH) = HIGH(MEBAS);
   RETURN;
   END INIT$FP;
```

```
/*****************************************************************
*
* ADJUST$OP:
*   PROCEDURE USED TO PUT TWO VECTORS INTO THE TABLE VECTOR.
*
* PARAMETERS:
*   - A,B.- POINTERS TO FIRST AND SECOND VECTOR VALUES.
*
*****************************************************************/
ADJUST$OP: PROCEDURE(A,B);
   DCL (A,B) ADDRESS,
       OP1 BASED A (4) BYTE,
       OP2 BASED B (4) BYTE,
       I BYTE;
   DO I = 0 TO LAST(OP1);
      RES$TABLE(I) = OP1(I);
      RES$TABLE(I + 4) = OP2(I);
      END;
   RETURN;
   END ADJUST$OP;
```

```
/*****************************************************************
*
* ADJUST1$OP:
*   PROCEDURE USED TO PUT ONE VECTOR INTO THE TABLE VECTOR.
*
* PARAMETERS:
*   -  A.- POINTER TO VECTOR VALUE.
*
*****************************************************************/
ADJUST1$OP: PROCEDURE (A);
   DCL A ADDRESS,
       OP1 BASED A (4) BYTE,
       I BYTE;
   DO I = 0 TO LAST(OP1);
      RES$TABLE(I) = OP1(I);
      END;
   RETURN;
   END ADJUST1$OP;
```

```
/*****************************************************************
*
* ADJUST2$OP:
*   PROCEDURE USED TO PUT TWO ADDRESS VALUES INTO THE TABLE VECTOR.
*
* PARAMETERS:
*   - A,B.- POINTERS TO TWO ADDRESS VALUES.
*
*****************************************************************/
ADJUST2$OP: PROCEDURE (A,B);
   DCL (A,B) ADDRESS,
       OP1 BASED A (2) BYTE,
       OP2 BASED B (2) BYTE,
       I BYTE;
   DO I = 0 TO LAST(OP1);
      RES$TABLE(I) = OP1(I);
      RES$TABLE(I + 4) = OP2(I);
      END;
   RETURN;
   END ADJUST2$OP;
```

```
/*****************************************************************
*
* VAL$RESULT:
*   PROCEDURE USED TO GET THE RESULT IN FIRST FOUR BYTES OF THE
*   TABLE VECTOR, AND PUT IT INTO ANOTHER VECTOR PROVIDED.
*
* PARAMETERS:
*   - C.- POINTER TO VECTOR ON WHICH RESULT IS DESIRED.
*
*****************************************************************/
VAL$RESULT: PROCEDURE(C);
   DCL C ADDRESS,
       RESULT BASED C (4) BYTE,
       I BYTE;
   DO I = 0 TO LAST(RESULT);
      RESULT(I) = RES$TABLE(I);
      END;
   RETURN;
   END VAL$RESULT;
```

```
/*****************************************************************
*
* VAL$RESULT$1:
*   PROCEDURE USED TO PUT ALL EIGHT BYTES OF THE TABLE VECTOR
*   INTO TWO FOUR BYTES VECTORS PROVIDED.
*
* PARAMETERS:
*   - A,B.- POINTERS TO VECTORS ON WHICH RESULT IS DESIRED. A
*     MUST POINT TO FIRST VECTOR (FIRST FOUR BYTES OF
*     TABLE), AND B MUST POINT TO THE SECOND VECTOR (LAST FOUR
*     BYTES IN TABLE).
*
*****************************************************************/
VAL$RESULT$1: PROCEDURE(A,B);
   DCL (A,B) ADDRESS,
       OP1 BASED B (4) BYTE,
       OP2 BASED A (4) BYTE,
       I BYTE;
   DO I = 0 TO LAST(OP1);
      OP2(I) = RES$TABLE(I);
      OP1(I) = RES$TABLE(I + 4);
      END;
   RETURN;
   END VAL$RESULT$1;
```

```
/*****************************************************************
*
* VAL$RESULT$2:
*   PROCEDURE USED TO GET THE RESULT FROM FIXED POINT
*   DIVISION OPERATION, AND PUT THEM INTO TWO ADDRESS
*   LOCATIONS PROVIDED.
*
* PARAMETERS:
*   - C,R.- POINTERS TO TWO ADDRESS LOCATIONS IN WHICH THE
*     RESULT IS DESIRED TO BE PLACED.
*
*****************************************************************/
VAL$RESULT$2: PROCEDURE (C,R);
   DCL (C,R) ADDRESS,
       OP1 BASED C (2) BYTE,
       OP2 BASED R (2) BYTE,
       I BYTE;
   DO I = 0 TO LAST(OP1);
      OP1(I) = RES$TABLE(I);
      OP2(I) = RES$TABLE(I + 4);
      END;
   END VAL$RESULT$2;
```

```
/*****************************************************************
*
* COMPARE:
*   PROCEDURE USED TO CHECK FOR OUTPUT CONDITIONS FROM F.P. BOARD.
*
* PARAMETERS:
*   - A.- BYTE VALUE CONTAINING RESULT FROM F.P. BOARD.
*   - B.- BYTE VALUE CONTAINING CONDITION DESIRED TO BE CHECKED:
*            <  ..... 0
*            <= ..... 1
*            >  ..... 2
*            >= ..... 3
*            =  ..... 4
*            <> ..... 5
* USAGE:
*   TYPED PROCEDURE THAT RETURNS A 'TRUE' VALUE (001H) IF OUTPUT
*   CONDITION FROM F.P. BOARD AND CONDITION DESIRED TO BE TESTED
*   ARE SIMILAR. IF NOT SIMILAR, A 'FALSE' VALUE (000H) IS RETURNED.
*
*****************************************************************/
COMPARE: PROCEDURE (A,B) BYTE;
   DCL TRUE LIT '01H',
       FALSE LIT '00H';
   DCL (A,B) BYTE,
       (LESS$THAN,LESS$OR$EQUAL,GREATER$THAN,GREAT$OR$EQUAL,EQUAL,NOT$EQUAL)
       BYTE DATA (0,1,2,3,4,5);
   IF ((A = 80H) AND ((B = LESS$OR$EQUAL) OR (B = GREAT$OR$EQUAL) OR
      (B = EQUAL))) THEN RETURN TRUE;
   IF ((A = 40H) AND ((B = GREATER$THAN) OR (B = GREAT$OR$EQUAL) OR
      (B = NOT$EQUAL))) THEN RETURN TRUE;
   IF ((A = 20H) AND ((B = LESS$THAN) OR (B = LESS$OR$EQUAL) OR
      (B = NOT$EQUAL))) THEN RETURN TRUE;
   RETURN FALSE;
   END COMPARE;
```

```
/*****************************************************************
*
* FLOAT$MSG$ERROR:
*   PROCEDURE USED TO SEND A MESSAGE ERROR ACCORDING TO ERROR
*   CODE PROVIDED BY F.P. BOARD.
*
* USAGE:
*   UNTYPED PROCEDURE. IF CONDITION OF ERROR IS DETECTED, THIS
*   PROCEDURE MUST BE CALLED IN ORDER TO OBTAIN ERROR CODE AND
*   DISPLAY AN APPROPRIATE MESSAGE. NOTE THAT PROGRAM EXECUTION
*   WILL BE STOPPED AND INTERRUPTS WILL BE ENABLED IF AN ERROR
*   IS DETECTED.
*
*****************************************************************/
FLOAT$MSG$ERROR: PROCEDURE;
   DCL MSG1(*) BYTE DATA ('DIVISION BY ZERO.$$'),
       MSG2(*) BYTE DATA ('DOMAIN ERROR.$$'),
       MSG3(*) BYTE DATA ('OVERFLOW.$$'),
       MSG4(*) BYTE DATA ('UNDERFLOW.$$'),
       MSG5(*) BYTE DATA ('INVALID FORMAT FOR FIRST ARGUMENT.$$'),
       MSG6(*) BYTE DATA ('INVALID FORMAT FOR SECOND ARGUMENT.$$'),
       ERROR(*) BYTE DATA (' FATAL ERROR.$$'),
       I BYTE;
   I = INPUT(IN$STATUS) AND 07H;
   DO CASE I;
      ;
```

```
        CALL CRT$PRINT$STRING(.MSG1);
        CALL CRT$PRINT$STRING(.MSG2);
        CALL CRT$PRINT$STRING(.MSG3);
        CALL CRT$PRINT$STRING(.MSG4);
        CALL CRT$PRINT$STRING(.MSG5);
        CALL CRT$PRINT$STRING(.MSG6);
        ;
        END;
    CALL CRT$PRINT$STRING(.ERROR);
    CALL SEND$CRLF;
    RETURN;
    END FLOAT$MSG$ERROR;
```

```
/*****************************************************************************
*
*
* CHECK:
*   PROCEDURE USED TO CHECK FOR STATUS OF F.P. BOARD AND TO DETECT IF
*   ANY ERROR HAS OCCURRED.
*
* USAGE:
*   UNTYPED PROCEDURE THAT IS CALLED BY ALL PROCEDURES THAT TRY TO EXECUTE
*   A FLOATING POINT OPERATION WITH THE F.P. BOARD.
*
*****************************************************************************/
CHECK: PROCEDURE;
   DCL I BYTE;
   DO WHILE ((I:=INPUT(IN$FLAG)) AND BUSY$MASK) = BUSY$MASK;
      END;
   IF (I AND ERROR$MASK) = ERROR$MASK
      THEN DO;
         CALL FLOAT$MSG$ERROR;
         HALT;
         END;
   RETURN;
   END CHECK;
```

```
/*****************************************************************************
*
* MUL:
*   PROCEDURE USED TO PERFORM FIXED POINT MULTIPLICATION USING THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - A.- POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE FIRST
*         OPERAND WILL BE LOCATED.
*   - B.- POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE SECOND
*         OPERAND WILL BE LOCATED.
*   - C.- POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE RESULT
*         IS DESIRED TO BE PLACED. NOTICE THAT IT COULD BE THE SAME
*         LOCATION USED FOR ANY OF THE OPERANDS.
*
*****************************************************************************/
MUL: PROCEDURE (A,B,C) PUBLIC;
   DCL (A,B,C) ADDRESS;
   CALL ADJUST2$OP (A,B);
   OUTPUT(OUT$OP$CODE) = MUL$CODE;
   CALL CHECK;
   CALL VAL$RESULT (C);
   RETURN;
   END MUL;
```

```
/*************************************************************************
*
* DIV:
*   PROCEDURE USED TO PERFORM FIXED POINT DIVISION USING THE
*   F.P. BOARD.
*
* PARAMETERS:
*   - A.- POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE DIVI-
*     DEND IS LOCATED.
*   - B.- POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE DIVI-
*     DER IS LOCATED.
*   - C.- POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE RESULT
*     (QUOTIENT) IS DESIRED TO BE PLACED.
*   - R.- POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE REMAIN-
*     DER IS DESIRED TO BE PLACED. NOTICE THAT C AND R COULD POINT TO ANY
*     OF THE TWO OPERANDS IF SO DESIRED.
*
*************************************************************************/
DIV: PROCEDURE (A,B,C,R) PUBLIC;
   DCL (A,B,C,R) ADDRESS;
   CALL ADJUST2$OP(A,B);
   OUTPUT(OUT$OP$CODE) = DIV$CODE;
   CALL CHECK;
   CALL VAL$RESULT$2(C,R);
   RETURN;
   END DIV;
```

```
/*****************************************************************************
*
* EDIV:
*   PROCEDURE USED TO PERFORM EXTENDED FIXED POINT DIVISION USING THE
*   F.P. BOARD.
*
* PARAMETERS:
*   - A.- POINTER TO A 4 BYTE VECTOR THAT WILL PROVIDE THE DIVIDEND.
*   - B.- POINTER TO A 2 BYTE VECTOR THAT WILL PROVIDE THE DIVIDER.
*   - C.- POINTER TO A 4 BYTE VECTOR IN WHICH THE QUOTIENT WILL BE RETURNED.
*   - R.- POINTER TO A 4 BYTE VECTOR IN WHICH THE REMAINDER WILL BE RETURNED.
*
*****************************************************************************/
EDIV: PROCEDURE (A,B,C,R) PUBLIC;
   DCL (A,B,C,R) ADDRESS,
       OP2 BASED B (2) BYTE,
       I BYTE;
   CALL ADJUST1$OP (A);
   DO I = 0 TO LAST(OP2);
      RES$TABLE(I + 4) = OP2(I);
      END;
   OUTPUT(OUT$OP$CODE) = EDIV$CODE;
   CALL CHECK;
   CALL VAL$RESULT1 (C,R);
   RETURN;
   END EDIV;
```

```
/*****************************************************************************
*
* FMUL:
*   PROCEDURE USED TO PERFORM FLOATING POINT MULTIPLICATION USING THE
*   F.P. BOARD.
*
* PARAMETERS:
*   -A,B,C.- POINTERS TO 3 FOUR BYTE VECTORS THAT POINT TO THE TWO OPERANDS
*    AND THE RESULT RESPECTIVELY. NOTE THAT THE RESULT COULD BE THE SAME
*    VECTOR USED TO INDICATE ANY OPERAND.
*
*****************************************************************************/
FMUL: PROCEDURE (A,B,C) PUBLIC;
   DCL (A,B,C) ADDRESS;
   CALL ADJUST$OP(A,B);
   OUTPUT(OUT$OP$CODE) = FMUL$CODE;
   CALL CHECK;
   CALL VAL$RESULT(C);
   RETURN;
   END FMUL;
```

```
/*************************************************************************
*
* FDIV:
*   PROCEDURE USED TO PERFORM FLOATING POINT DIVISION USING THE F.P BOARD.
*
* PARAMETERS:
*   - A,B,C.- POINTERS TO 3 FOUR BYTE VECTORS THAT WILL POINT TO THE DIVIDEND
*     AND THE DIVISOR ON THE FIRST TWO, AND THE RESULT ON THE THIRD. NOTE
*     THAT THE RESULT COULD BE PUT IN THE SAME PLACE OF ANY OPERAND IF SO
*     DESIRED.
*
*************************************************************************/
FDIV: PROCEDURE (A,B,C) PUBLIC;
   DCL (A,B,C) ADDRESS;
   CALL ADJUST$OP (A,B);
   OUTPUT(OUT$OP$CODE) = FDIV$CODE;
   CALL CHECK;
   CALL VAL$RESULT(C);
   RETURN;
   END FDIV;
```

```
/*****************************************************************************
*
* FADD:
*   PROCEDURE USED TO PERFORM FLOATING POINT ADDITION USING THE F.P. BOARD.
*
* PARAMETERS:
*   - A,B,C.- POINTERS TO 3 FOUR BYTE VECTORS. THE FIRST TWO POINT TO THE
*     TWO OPERANDS, AND THE THIRD ONE POINTS TO THE VECTOR RESULT. NOTE
*     THAT THE RESULT COULD ALSO BE PLACED IN ANY OF THE OPERAND VECTORS.
*
*****************************************************************************/
FADD: PROCEDURE (A,B,C) PUBLIC;
   DCL (A,B,C) ADDRESS;
   CALL ADJUST$OP (A,B);
   OUTPUT(OUT$OP$CODE) = FADD$CODE;
   CALL CHECK;
   CALL VAL$RESULT (C);
   RETURN;
   END FADD;
```

```
/*****************************************************************************
*
* FSUB:
*   PROCEDURE USED TO PERFORM FLOATING POINT SUBTRACTION USING THE
*   F.P. BOARD.
*
* PARAMETERS:
*   - A,B,C.- POINTERS TO 3 FOUR BYTE VECTORS. THE FIRST TWO POINT TO
*     BOTH OPERANDS, AND THE THIRD ONE POINTS TO THE VECTOR WHERE THE
*     RESULT IS DESIRED TO BE PLACED. NOTE THAT THE RESULT COULD BE
*     PLACED IN ANY OF BOTH OPERANDS.
*
*****************************************************************************/
FSUB: PROCEDURE (A,B,C) PUBLIC;
   DCL (A,B,C) ADDRESS;
   CALL ADJUST$OP (A,B);
   OUTPUT(OUT$OP$CODE) = FSUB$CODE;
   CALL CHECK;
   CALL VAL$RESULT (C);
   RETURN;
   END FSUB;
```

```
/*****************************************************************************
*
* FSQR:
*   PROCEDURE USED TO PERFORM A FLOATING POINT SQUARE USING THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - A,C.- POINTERS TO 2 FOUR BYTE VECTORS. A POINTS TO A VECTOR IN WHICH
*     A FLOATING POINT QUANTITY IS LOCATED AND TO WHICH ITS SQUARE WILL BE
*     OBTAINED. C POINTS TO A VECTOR IN WHICH THE RESULT IS DESIRED TO BE
*     PLACED. NOTE THAT A AND C COULD POINT TO THE SAME VECTOR.
*
*****************************************************************************/
FSQR: PROCEDURE (A,C)  PUBLIC;
   DCL (A,C) ADDRESS;
   CALL ADJUST1$OP(A);
   OUTPUT(OUT$OP$CODE) = FSQR$CODE;
   CALL CHECK;
   CALL VAL$RESULT (C);
   RETURN;
   END FSQR;
```

```
/*****************************************************************************
*
* FLTDS:
*   PROCEDURE USED TO PERFORM A FIXED-TO-FLOAT CONVERSION USING THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - A,C.- POINTERS TO 2 FOUR BYTE VECTORS. A POINTS TO A VECTOR CONTAINING
*     A FIXED POINT INTEGER AND C POINTS TO A VECTOR WHERE THE SINGLE PRE-
*     CISION FLOATING POINT REPRESENTATION OF THE SAME VALUE, IS DESIRED TO
*     BE PLACED. NOTE THAT BOTH, A AND C, COULD POINT TO THE SAME VECTOR.
*
*****************************************************************************/
FLTDS: PROCEDURE (A,C) PUBLIC;
   DCL (A,C) ADDRESS;
   CALL ADJUST1$OP (A);
   OUTPUT(OUT$OP$CODE) = FLTDS$CODE;
   CALL CHECK;
   CALL VAL$RESULT (C);
   RETURN;
   END FLTDS;
```

```
/*****************************************************************************
*
* FIXSD:
*   PROCEDURE USED TO PERFORM A FLOAT-TO-FIXED CONVERSION USING THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - A,C.- POINTERS TO 2 FOUR BYTE VECTORS. A POINTS TO A VECTOR CONTAINING
*     THE FIXED POINT QUANTITY DESIRED TO BE CONVERTED. C POINTS TO A VECTOR
*     IN WHICH THE CONVERSION IS DESIRED TO BE PLACED. NOTE THAT BOTH COULD
*     POINT TO THE SAME VECTOR.
*
*****************************************************************************/
FIXSD: PROCEDURE (A,C) PUBLIC;
   DCL (A,C) ADDRESS;
   CALL ADJUST1$OP (A);
   OUTPUT(OUT$OP$CODE) = FIXSD$CODE;
   CALL CHECK;
   CALL VAL$RESULT (C);
   RETURN;
   END FIXSD;
```

```
/*****************************************************************************
*
* FSQRT:
*   PROCEDURE TO PERFORM THE SQUARE ROOT OF A FLOATING POINT NUMBER USING
*   THE F.P. BOARD.
*
* PARAMETERS:
*   - A,C.- POINTERS TO 2 FOUR BYTE VECTORS. A POINTS TO THE VECTOR CONTAINING
*     THE NUMBER TO WHICH ITS SQUARE ROOT WILL BE OBTAINED. C POINTS TO THE
*     VECTOR IN WHICH THE RESULT WILL BE PLACED. NOTE THAT BOTH POINTERS
*     COULD BE REFERING TO THE SAME VECTOR.
*
*****************************************************************************/
FSQRT: PROCEDURE (A,C)  PUBLIC;
   DCL (A,C) ADDRESS;
   CALL ADJUST1$OP (A);
   OUTPUT(OUT$OP$CODE) = FSQRT$CODE;
   CALL CHECK;
   CALL VAL$RESULT (C);
   RETURN;
   END FSQRT;
```

```
/*****************************************************************************
*
* FCMPR:
*   PROCEDURE USED TO COMPARE TWO FLOATING POINT NUMBERS USING THE F.P. BOARD.
*
* PARAMETERS:
*   - A,B.- POINTERS TO 2 FOUR BYTE VECTORS CONTAINING THE TWO FLOATING
*     POINT NUMBERS DESIRED TO BE COMPARED.
*   - C.- POINTS TO A BYTE VALUE CONTAINING THE CODE CORRESPONDING TO THE
*     TYPE OF COMPARISON DESIRED:
*                    <  ..... 0
*                    <= ..... 1
*                    >  ..... 2
*                    >= ..... 3
*                    =  ..... 4
*                    <> ..... 5
*
* USAGE:
*   TYPED PROCEDURE. IF THE RELATION DESIRED TO BE TESTED HOLDS, A VALUE OF
*   '01H' (TRUE) IS RETURNED, OTHERWISE A VALUE OF '00H' (FALSE) WILL BE
*   RETURNED.
*
*****************************************************************************/
FCMPR: PROCEDURE (A,B,C) BYTE PUBLIC;
   DCL (A,B,C) ADDRESS,
       (TEST BASED C,RESULT, FLAG) BYTE;
   FLAG = 00H;          /* RESET FLAG TO CHECK FOR TWO NEGATIVES NUMBERS */
   CALL ADJUST$OP (A,B);
   IF (RES$TABLE(3) >= 80H) AND
      (RES$TABLE(7) >= 80H)
      THEN FLAG = 0FFH;
```

```
OUTPUT(OUT$OP$CODE) = FCMPR$CODE;
CALL CHECK;
RESULT = INPUT (IN$STATUS) AND 0E0H;
IF FLAG AND (RESULT <> 80H)
THEN DO;
     IF RESULT = 40H
        THEN RESULT = 20H;
        ELSE RESULT = 40H;
     END;
RETURN (RESULT:= COMPARE(RESULT,TEST));
END FCMPR;
```

```
/*****************************************************************************
*
* FZTST:
*   PROCEDURE USED TO TEST A FLOATING POINT NUMBER AGAINST 0.0 USING THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - A.- POINTER TO A FOUR BYTE VECTOR CONTAINING THE FLOATING POINT VALUE
*     DESIRED TO BE TESTED.
*   - C.- POINTER TO A BYTE VALUE CONTAINING THE CODE OF THE COMPARISON DESI-
*     RED, ACCORDING TO THE FOLLOWING RULES:
*                    <  ..... 0
*                    <= ..... 1
*                    >  ..... 2
*                    >= ..... 3
*                    =  ..... 4
*                    <> ..... 5
*
* USAGE:
*   TYPED PROCEDURE. IF THE RELATION DESIRED TO BE TESTED HOLDS, A VALUE OF
*   '01H' (TRUE) IS RETURNED, OTHERWISE A VALUE OF '00H' (FALSE) WILL BE RE-
*   TURNED.
*
*****************************************************************************/
FZTST: PROCEDURE (A,C) BYTE PUBLIC;
   DCL (A,C) ADDRESS,
       (TEST BASED C,RESULT) BYTE;
   CALL ADJUST1$OP (A);
   OUTPUT(OUT$OP$CODE) = FZTST$CODE;
   CALL CHECK;
   RESULT = INPUT (IN$STATUS) AND 0E0H;
   RETURN (RESULT:= COMPARE(RESULT,TEST));
   END FZTST;
```

```
/*****************************************************************************
*
* EXCH:
*   PROCEDURE USED TO EXCHANGE TWO FLOATING POINT VALUES USING THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - A,C.- POINTERS TO 2 FOUR BYTE VECTORS CONTAINING TWO FLOATING
*     POINT NUMBERS DESIRED TO BE EXCHANGED.
*
*****************************************************************************/
EXCH: PROCEDURE (A,C) PUBLIC;
   DCL (A,C) ADDRESS;
   CALL ADJUST$OP (A,C);
   OUTPUT(OUT$OP$CODE) = EXCH$CODE;
   CALL CHECK;
   CALL VAL$RESULT$1(A,C);
   RETURN;
   END EXCH;
```

```
/*****************************************************************************
*
* COS$SIN:
*   THIS PROCEDURE IS USED TO CALCULATE THE COSINE AND SINE FUNCTIONS
*   OF A GIVEN ARGUMENT IN RADIANS.
*
* PARAMETERS:
*   - A.- POINTER TO A FOUR BYTE VECTOR IN WHICH THE FLOATING POINT REPRE-
*     SENTATION OF AN ANGLE IN RADIANS IS LOCATED.
*
*   - C.- POINTER TO A FOUR BYTE VECTOR IN WHICH THE VALUE OF THE COSINE
*     OF THE GIVEN ANGLE, WILL BE PLACED.
*
*   - S.- POINTER TO A FOUR BYTE VECTOR IN WHICH THE VALUE OF THE SINE OF
*     THE GIVEN ANGLE WILL BE PLACED.
*
*****************************************************************************/
COS$SIN: PROCEDURE (A,C,S) PUBLIC;
   DCL (A,C,S) ADDRESS,
       ANGLE BASED A (4) BYTE,
       COSINE BASED C (4) BYTE,
       SINE BASED S (4) BYTE.
```

```
ANGLE$SQ (4) BYTE,
TEMP (4) BYTE,
TEMP0 (4) BYTE,
TEMP1 (4) BYTE,
MINUS$ONE (4) BYTE DATA (00H,00H,80H,0BFH),    /* -1.0 */
ONE$FLOAT (4) BYTE DATA (00H,00H,80H,3FH),     /* 1.0 IN F.P. FORMAT.*/
TWO$FLOAT (4) BYTE DATA (00H,00H,00H,40H),     /* 2.0 */
PI$FLOAT (4) BYTE DATA (0DBH,0FH,49H,40H),     /* 3.141593 */
TWO$PI (4) BYTE DATA (0DBH,0FH,0C9H,40H),      /* 6.2831853 */
PI$OVER2 (4) BYTE DATA (0DBH,0FH,0C9H,3FH),    /* 1.5707963 */
PI$3$OVER2 (4) BYTE DATA (0E4H,0CBH,96H,40H), /* 4.7123889 */
CONST$1 (4) BYTE DATA (0B5H,1FH,12H,3FH),      /* 0.5707963 */
CONST$2 (4) BYTE DATA (0E6H,5DH,25H,0BFH),     /* -0.645964 */
CONST$3 (4) BYTE DATA (0E1H,35H,0A3H,3DH),     /* 0.079629261 */
CONST$4 (4) BYTE DATA (0AAH,68H,99H,0BBH),     /* -0.0046816668 */
CONST$5 (4) BYTE DATA (25H,0BH,28H,39H),       /* 0.0001602588 */
CONST$6 (4) BYTE DATA (0A4H,067H,66H,0B6H),    /* -0.0000034333 */
CHECK BYTE DATA (002H),    /* CHECK FOR GREATER THAN */
CHECK1 BYTE DATA (003H),   /* CHECK FOR GREATER THAN OR EQUAL */
CHECK2 BYTE DATA (001H),   /* CHECK FOR LESS THAN OR EQUAL */
CHECK3 BYTE DATA (004H),   /* CHECK FOR EQUAL */
(SIGN,SIGN1,QUAD,I) BYTE;
```

```
   DO I = 0 TO 3;
      TEMP0(I) = ANGLE(I);
      END;
/*    CHECK IF ANGLE IS >= 360 DEGREES.    */
   SIGN = FCMPR(.TEMP0,.TWO$PI,.CHECK1);
   DO WHILE SIGN;
      CALL FSUB(.TEMP0,.TWO$PI,.TEMP0);
      SIGN = FCMPR(.TEMP0,.TWO$PI,.CHECK1);
      END;
/*    CHECK IF ANGLE IS NEGATIVE.    */
   DO WHILE TEMP0(3) >= 080H;
      CALL FADD(.TEMP0,.TWO$PI,.TEMP0);
      END;
/*    CHECK FOR SPECIAL CASES    */
   IF FCMPR(.TEMP0, .PI$OVER2, .CHECK3)
   THEN DO;     /* 90 DEGREES */
        DO I = 0 TO 3;
           SINE(I) = ONE$FLOAT(I);
           END;
        RETURN;
        END;
   IF FCMPR(.TEMP0, .PI$3$OVER2, .CHECK3)
   THEN DO;     /* 270 DEGREES */
        DO I = 0 TO 3;
           SINE(I) = MINUS$ONE(I);
           END;
        RETURN;
```

```
        END;
    IF FCMPR(.TEMP0, .TWO$PI, .CHECK3)
    THEN DO;    /* 360 DEGREES */
        DO I = 0 TO 3;
            COSINE(I) = ONE$FLOAT(I);
            END;
        RETURN;
        END;
/*   TO NORMALIZE THE ANGLE BETWEEN 0 AND 90 DEGREES.   */
    QUAD = 1;
    IF (SIGN := FCMPR(.TEMP0,.PI$OVER2,.CHECK)) AND
       (SIGN1 := FCMPR(.TEMP0,.PI$FLOAT,.CHECK2))
    THEN DO;
       QUAD = 2;
       CALL FSUB(.PI$FLOAT,.TEMP0,.TEMP0);
       END;
    ELSE DO;
       IF (SIGN := FCMPR(.TEMP0,.PI$FLOAT,.CHECK)) AND
          (SIGN1 := FCMPR(.TEMP0,.PI$3$OVER2,.CHECK2))
       THEN DO;
          QUAD = 3;
          CALL FSUB(.TEMP0,.PI$FLOAT,.TEMP0);
          END;
       ELSE DO;
          IF (SIGN := FCMPR(.TEMP0,.PI$3$OVER2,.CHECK))
          THEN DO;
             QUAD = 4;
             CALL FSUB(.TWO$PI,.TEMP0,.TEMP0);
             END;
          END;
```

```
      END;
/*    CONVERT ANGLE IN RADIANS TO ANGLE IN SEMICIRCLE UNITS.    */
   CALL FDIV(.TEMP0,.PI$FLOAT,.TEMP);
/*    GET THE SQUARE OF THE ANGLE.    */
   CALL FSQR(.TEMP,.ANGLE$SQ);
/*    PERFORM HASTINGS APPROXIMATION.    */
   CALL FMUL(.ANGLE$SQ,.CONST$6,.TEMP1);
   CALL FADD(.TEMP1,.CONST$5,.TEMP1);
   CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
   CALL FADD(.TEMP1,.CONST$4,.TEMP1);
   CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
   CALL FADD(.TEMP1,.CONST$3,.TEMP1);
   CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
   CALL FADD(.TEMP1,.CONST$2,.TEMP1);
   CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
   CALL FADD(.TEMP1,.CONST$1,.TEMP1);
   CALL FMUL(.TEMP,.TEMP1,.TEMP1);
   CALL FADD(.TEMP1,.TEMP,.TEMP1);
   CALL FSQR(.TEMP1,.TEMP1);
   CALL FMUL(.TEMP1,.TWO$FLOAT,.TEMP1);
/*    TO COMPUTE THE VALUE OF THE COSINE.    */
   CALL FSUB(.ONE$FLOAT,.TEMP1,.COSINE);
/*    TO COMPUTE THE VALUE OF THE SINE.    */
   CALL FSQR(.COSINE,.TEMP1);
   CALL FSUB(.ONE$FLOAT,.TEMP1,.TEMP1);
   CALL FSQRT(.TEMP1,.SINE);
/*    GIVE SIGNS TO COSINE AND SINE VALUES ACCORDING TO QUADRANT    */
   DO CASE QUAD;
```

```
      ;
      ;                                          /*   FIRST QUADRANT   */
      COSINE(3) = COSINE(3) XOR 80H;             /*   SECOND QUADRANT   */
      DO;
         COSINE(3) = COSINE(3) XOR 80H;          /*   THIRD QUADRANT   */
         SINE(3) = SINE(3) XOR 80H;
      END;
      SINE(3) = SINE(3) XOR 80H;                 /*   FOURTH QUADRANT   */
   END;
/*   ALL DONE. RETURN.   */
   END COS$SIN;
```

```
/*****************************************************************************
*
* ARC$TAN:
*   THIS PROCEDURE IS USED TO CALCULATE THE ARC$TAN FUNCTION IN RADIANS
*   GIVEN AS ARGUMENT A RATIO OF TWO VALUES IN FP REPRESENTATION.
*
* PARAMETERS:
*   - X.- POINTER TO A FOUR BYTE VECTOR REPRESENTING THE DENOMINATOR
*     OF THE RATIO.
*   - Y.- POINTER TO A FOUR BYTE VECTOR REPRESENTING THE NUMERATOR
*     OF THE RATIO.
*   - A.- POINTER TO A FOUR BYTE VECTOR IN WHICH THE VALUE OF THE ANGLE
*     (IN RADIANS) WILL BE PLACED AFTER CALCULATION OF THE ARC$TAN.
*
*****************************************************************************/
ARC$TAN: PROCEDURE (X,Y,A) PUBLIC;
   DCL (X,Y,A) ADDRESS,
       DELTA$X BASED X (4) BYTE,
       DELTA$Y BASED Y (4) BYTE,
       ANGLE BASED A (4) BYTE,
       PI$FLOAT (4) BYTE DATA (0DBH,0FH,49H,40H),        /* 3.141593 */
       TWO$PI (4) BYTE DATA (0DBH,0FH,0C9H,40H),         /* 6.2831853 */
       PI$OVER4 (4) BYTE DATA (0DBH,0FH,49H,3FH),        /* 0.78539819 */
       PI$OVER2 (4) BYTE DATA (0DBH,0FH,0C9H,3FH),       /* 1.5707963 */
       PI$3$OVER2 (4) BYTE DATA (0E4H,0CBH,96H,40H),     /* 4.7123889 */
       CONST$1 (4) BYTE DATA (0F5H,0FFH,7FH,3FH),        /* 0.9999993329 */
       CONST$2 (4) BYTE DATA (1CH,0A6H,0AAH,0BEH),       /* -0.3332985605 */
       CONST$3 (4) BYTE DATA (0A7H,40H,4CH,3EH),         /* 0.1994653599 */
       CONST$4 (4) BYTE DATA (63H,6CH,0EH,0BEH),         /* -0.1390853351 */
       CONST$5 (4) BYTE DATA (0DEH,77H,0C5H,3DH),        /* 0.0964200441 */
       CONST$6 (4) BYTE DATA (0C4H,01H,65H,0BDH),        /* -0.0559098861 */
       CONST$7 (4) BYTE DATA (51H,16H,0B3H,3CH),         /* 0.0218612288 */
       CONST$8 (4) BYTE DATA (0E6H,0D7H,84H,0BBH),       /* -0.0040540580 */
       CHECK BYTE DATA (04H),
```

```
        MSG1(*) BYTE DATA
              (' ARC$TAN FUNCTION UNDEFINED FOR BOTH ARGUMENTS EQUAL TO ZERO.$$'),
        MSG2(*) BYTE DATA (' FATAL ERROR.$$'),
        Z (4) BYTE,
        Z$SQUARE (4) BYTE,
        TEMP (4) BYTE,
        TEMP1 (4) BYTE,
        (SIGN$X, SIGN$Y, ZERO$X, ZERO$Y, I) BYTE;
    SIGN$X, SIGN$Y = 00H;
    DO I = 0 TO 3;
       TEMP(I) = DELTA$Y(I);
       TEMP1(I) = DELTA$X(I);
       END;
/*    SAVE SIGN TO DETERMINE QUADRANT    */
    IF TEMP(3) >= 80H THEN SIGN$Y = 0FFH;
    IF TEMP1(3) >= 80H THEN SIGN$X = 0FFH;
/*    CHECK FOR VALID ARGUMENTS    */
    IF (ZERO$Y := FZTST(.DELTA$Y,.CHECK)) AND
       (ZERO$X := FZTST(.DELTA$X,.CHECK))
    THEN DO;
         CALL CRT$PRINT$STRING (.MSG1);
         CALL CRT$PRINT$STRING (.MSG2);
         HALT;
         END;
    IF ZERO$X
    THEN DO;
         IF SIGN$Y
         THEN DO;
              DO I = 0 TO 3;
                 ANGLE(I) = PI$3$OVER2(I);
                 END;
              RETURN;
              END;
```

```
        ELSE DO;
             DO I = 0 TO 3;
                ANGLE(I) = PI$OVER2(I);
                END;
             RETURN;
             END;
        END;
/*   FORM Z TO PERFORM HASTINGS APPROXIMATION   */
   TEMP(3) = TEMP(3) AND 7FH;
   TEMP1(3) = TEMP1(3) AND 7FH;
   CALL FSUB(.TEMP, .TEMP1, .Z);
   CALL FADD(.TEMP, .TEMP1, .TEMP);
   CALL FDIV(.Z, .TEMP, .Z);
   CALL FSQR(.Z, .Z$SQUARE);
/*   PERFORM HASTINGS APPROXIMATION FOR ARC$TAN   */
   CALL FMUL(.Z$SQUARE, .CONST$8, .TEMP);
   CALL FADD(.TEMP, .CONST$7, .TEMP);
   CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
   CALL FADD(.TEMP, .CONST$6, .TEMP);
   CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
   CALL FADD(.TEMP, .CONST$5, .TEMP);
   CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
   CALL FADD(.TEMP, .CONST$4, .TEMP);
   CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
   CALL FADD(.TEMP, .CONST$3, .TEMP);
   CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
   CALL FADD(.TEMP, .CONST$2, .TEMP);
   CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
   CALL FADD(.TEMP, .CONST$1, .TEMP);
   CALL FMUL(.Z, .TEMP, .TEMP);
   CALL FADD(.TEMP, .PI$OVER4, .ANGLE);
```

```
/*   RESTORE ANGLE TO PROPER QUADRANT    */
   IF (NOT SIGN$Y) AND SIGN$X              /* SECOND QUADRANT */
      THEN CALL FSUB(.PI$FLOAT, .ANGLE, .ANGLE);
   IF SIGN$Y AND SIGN$X                    /* THIRD QUADRANT */
      THEN CALL FADD(.PI$FLOAT, .ANGLE, .ANGLE);
   IF SIGN$Y AND (NOT SIGN$X)              /* FOURTH QUADRANT */
      THEN CALL FSUB(.TWO$PI, .ANGLE, .ANGLE);
/*   ALL DONE. RETURN.    */
   END ARC$TAN;



END FLOATING$POINT;
```

LIST OF REFERENCES

1. SBC 310 , High-Speed Mathematics Unit Hardware Reference Manual, Intel Corporation, Santa Clara, California, 1977.

2. 8080/8085 Flotaing Point Arithmetic Library User's Manual, Intel Corporation, Santa Clara, California, 1979.

3. The AM 9511 Arithmetic Processing Unit, Advanced Micro Devices, Sunnyvale, California, 1978.

4. Intellec Microcomputer Development System Hardware Reference Manual, Intel Corporation, Santa Clara, California, 1976.

5. ISIS-II PL/M Compiler Operator's Manual, Intel Corporation, Santa Clara, California, 1977.

6. PL/M-80 Programming Manual, Intel Corporation, Santa Clara, California, 1977.

7. ISIS-II System User's Guide, Intel Corporation, Santa Clara, California, 1977.

8. Skolnik, M.I., Introduction to Radar Systems, M.C. McGraw Hill, 1962.

9. NAVSHIPS 91921, 32(A), Performance Standard Sheet for Radar Set AN/SPS-10 (Series), 17 Apr. 1963.

10. NAVSHIPS 94120, Complimentary Technical Manual for AN/SPS-10E, 13 June 1961.

11. Wilson, C.H., Surface Search Radar Tracking by a Microcomputer Kalman Filter, M.S. Thesis, Naval Postgraduate School, Monterey, California, 1976.

12. Palmer, John F., The Intel Standard for Floating-Point Arithmetic, Intel Corporation, Santa Clara, California 1974.

13. Costella, F.R., and Dunnebache, F.G., Analytic Results for the X,Y-Kalman Tracking Filter, Johns Hopkins University, Silver Springs MD 20910. IEEE Transactions on Aerospace and Electronic Systems, Nov. 1974.

14. MCS-80 User's Manual, Intel Corporation, Santa Clara, California, 1977.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Professor D.E. Kirk, Code 62Ki Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	2
4.	Professor M.L. Cotton, Code 62Cc Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
5.	T.C. Deniz Kuvvetleri Komutanligi Egitim Sube Bakanliklar - ANKARA TURKEY	5
6.	Dz. Yzb. Fatih Erdogdu Yavuzselim Mevkufatci Sokak No 8/1 Fatih/ISTANBUL - TURKEY	1